UNIVERSIDAD DE LA RIOJA

# TRABAJO FIN DE ESTUDIOS

Título

## Interpretación de teoremas que sustentan técnicas de aprendizaje automático


Autor/es

GONZALO SANTAMARÍA GÓMEZ

Director/es

CÉSAR DOMÍNGUEZ PÉREZ y FRANCISCO JAVIER PÉREZ LÁZARO

Facultad

Facultad de Ciencia y Tecnología


Titulación

Grado en Matemáticas


Departamento

MATEMÁTICAS Y COMPUTACIÓN


Curso académico

2019-20



publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es


**UNIVERSIDAD DE LA RIOJA**

**Facultad de Ciencia y Tecnología**


# TRABAJO FIN DE GRADO

## Grado en Matemáticas

# Interpretación de teoremas que sustentan técnicas de aprendizaje automático


Realizado por:

**Gonzalo Santamaría Gómez**

Tutelado por:

**César Domínguez Pérez**

**Francisco Javier Pérez Lázaro**



**24 de junio de 2020**

# Resumen

En este trabajo vamos a estudiar desde el punto de vista matemático algunas de las técnicas usadas en modelos de aprendizaje automático. En concreto, nos centraremos en el concepto de neurona y de red neuronal.

Las neuronas tienen una serie de parámetros que podremos ajustar para que modelen ciertas funciones. Las redes neuronales son simplemente un conjunto de neuronas conectadas de cierta forma, los parámetros de una red neuronal son el conjunto de todos los parámetros de cada neurona.

Veremos que somos capaces de utilizar las redes neuronales para aproximarnos a cierto tipo de funciones y explicaremos cómo la red consigue reajustar sus parámetros de forma autónoma para mejorar la aproximación.

Este reajuste de parámetros se produce a través de un algoritmo de minimización de cierto error cometido. El problema de estos algoritmos es que no nos garantizan la convergencia a un mínimo global de la función error, por lo que nuestro objetivo será dar una serie de garantías que nos permitan decir cuándo un punto crítico es un mínimo global.

# Abstract

In this work we are going to study from the mathematical point of view some of the techniques used in machine learning models. Specifically, we will focus on the concept of neuron and neural network.

Neurons have a series of parameters that we can adjust to model certain functions. Neural networks are simply a set of neurons connected in a certain way, the parameters of a neural network are the set of all the parameters of each neuron.

We will see that we are able to use neural networks to approximate certain types of functions and we will explain how the network manages to readjust its parameters autonomously to improve the approximation.

This readjustment of parameters occurs through an algorithm of minimization of a certain error made. The problem with these algorithms is that they do not guarantee convergence to a global minimum of the error function, so our objective will be to provide a series of guarantees that allow us to say when a critical point is a global minimum.

# Introducción

Vamos a hacer un pequeño recorrido sobre algunas de las teorías matemáticas que respaldan el funcionamiento de las **redes neuronales**, un modelo matemático muy utilizado en los problemas de **aprendizaje automático** (*Machine Learning* en inglés), que es a su vez una rama de la **inteligencia artificial**.

Los modelos usados en inteligencia artificial tienen como objetivo ser capaces de realizar correctamente tareas que a los humanos nos parecen rutinarias, tales como el reconocimiento de personas o cualquier otra cosa en una imagen, interpretación y traducción de textos, recomendación de cierto contenido en una página web relacionado con tus búsquedas previas, etc. Antes, este tipo de tareas eran muy complejas para los ordenadores.

Las redes neuronales comenzaron a utilizarse con más frecuencia a partir de la última década ya que demostraron un mayor potencial para tratar con una gran cantidad de datos, porque son útiles en muchas y diversas situaciones y por el auge de el **aprendizaje profundo** (*Deep Learning* en inglés), una rama de el aprendizaje automático. Estos modelos pueden ser muy complejos y se pueden ver como un conjunto de operaciones organizadas de cierta forma para que, dada una **entrada** (foto, texto, vector de datos...) devuelva una **salida** (persona, traducción, objeto favorito de los clientes...). De todas las operaciones existentes nos vamos a quedar solamente con una y vamos a construir la **red neuronal hacia delante**, la cual está formado por neuronas.

La neurona matemática se puede ver como una función de $\mathbb{R}^n$ en $\mathbb{R}$, la cual tiene unos parámetros o **pesos** que se van a poder reajustar para devolver una salida adecuada.

En el Capítulo 1 construiremos el modelo matemático de red neuronal hacia delante, el cual se puede ver como una función de $\mathbb{R}^n$ en $\mathbb{R}^m$ con unos parámetros reajustables. Estos modelos nos van a servir para aproximarnos a esas hipotéticas[1] funciones $g^* : \mathbb{R}^n \to \mathbb{R}^m$ que, dada una entrada, devuelven una salida (dado un texto en Español devolver su traducción en Inglés). Como matemáticos nos podemos hacer la siguiente pregunta: ¿De verdad estos modelos son útiles? y si es así ¿Qué tipo de funciones vamos a poder aproximar? Estas preguntas las responderemos en el Capítulo 2.

[1]Son hipotéticas ya que es imposible saber a ciencia cierta cómo son estas funciones. Solo conocemos una pequeña parte de ellas (las entradas y salidas que disponemos).

Una vez hayamos obtenido una serie de resultados que nos garanticen una correcta aproximación, tenemos una segunda pregunta que responder. Sabemos que la red neuronal va a tener unos parámetros que nos van a permitir asemejarnos a $g^*$ tanto como queramos, pero ¿Cómo los podemos calcular? Esta pregunta la responderemos en el Capítulo 3.

A grandes rasgos, el método de aproximación de la red consiste en aplicarle un proceso de reajuste de pesos, el cual se realiza automáticamente a través de un método iterativo que minimiza el error cometido por la red en las entradas (por ejemplo, si para la entrada -hola- se devuelve la salida -wave-, la red está cometiendo un error en esa entrada el cual podremos hacer más pequeño hasta que devuelva -hello-).

Uno de los principales problemas que nos encontramos al aplicar el reajuste de pesos es la posible convergencia a mínimos locales o similares, ya que de esta manera no conseguimos obtener el valor más pequeño en nuestro error. En el Capítulo 4 explicaremos una forma de saber cuándo un punto crítico es un mínimo global de la función error.

# Índice general

# Capítulo 1

# Red neuronal multicapa

En este capítulo vamos construir un modelo de red neuronal multicapa, de forma muy simplificada y cuyo objetivo será aproximar funciones $g^* : \mathbb{R}^n \to \mathbb{R}^m$. También comentaremos los conceptos más básicos del aprendizaje automático y así poder entender mejor en las siguientes secciones el por qué de las estrategias usadas para aproximarnos a $g^*$ son de una determinada forma.

## 1.1. $\mathbb{R}^n$: espacio vectorial

**Definición 1.1** (Norma)**.** *Sea $\mathbb{R}$ el cuerpo de los números reales y sea $\mathcal{V}$ un espacio vectorial sobre $\mathbb{R}$. Una norma es una aplicación $\|\cdot\| : \mathcal{V} \to [0, \infty)$ satisfaciendo:*

1. $\|x + y\| \leq \|x\| + \|y\|$ $\forall x, y \in \mathcal{V}$ *(desigualdad triangular).*
2. $\|\lambda x\| = |\lambda| \|x\|$ $\forall x \in \mathcal{V}$ $y$ $\forall \lambda \in \mathbb{R}$.
3. $\|x\| = 0 \Rightarrow x = 0$.

*El par $(\mathcal{V}, \|\cdot\|)$ se dice que es un espacio vectorial normado.*

**Ejemplo 1.1** (Normas de $\mathbb{R}^n$)**.** *Consideramos el espacio vectorial $\mathcal{V} = \mathbb{R}^n$ con $n \in \mathbb{N}$. Tenemos las siguientes normas:*

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}\,; \quad p \in [1, \infty)$$

$$\|x\|_\infty = \sup_{i \leq n} \{|x_i|\}$$

$\forall x \in \mathbb{R}^n$.

Se puede demostrar que en $\mathbb{R}^n$ (o en cualquier espacio vectorial de dimensión finita) todas las normas son equivalentes (mirar [1]), esto es: si $\|\cdot\|_a$ y $\|\cdot\|_b$ son dos normas, existen $C_{a,b}$ y $C'_{a,b}$ constantes positivas tales que:

$$C_{a,b}\|x\|_a \leq \|x\|_b \leq C'_{a,b}\|x\|_a, \ \ \forall x \in \mathbb{R}^n$$

**Definición 1.2** (Producto escalar)**.** *Sea $\mathcal{V}$ un espacio vectorial sobre $\mathbb{R}$. Un producto escalar es una aplicación $\langle\cdot,\cdot\rangle : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ satisfaciendo:*

1. $\langle x, y\rangle = \langle y, x\rangle$, $\forall x, y \in \mathcal{V}$.
2. $\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$, $\forall x, y, z \in \mathcal{V}$.
3. $\langle \alpha x, y\rangle = \alpha\langle x, y\rangle$, $\forall x, y \in \mathcal{V}$ *y* $\forall \alpha \in \mathbb{R}$.
4. $\langle x, x\rangle \geq 0$, $\forall x \in \mathcal{V}$.
5. $\langle x, x\rangle = 0 \Rightarrow x = 0$, $\forall x \in \mathcal{V}$.

*El par $(\mathcal{V}, \langle\cdot,\cdot\rangle)$ se dice que es un espacio vectorial con producto escalar.*

**Ejemplo 1.2.** *Tomamos $\mathcal{V} = \mathbb{R}^n$, $n \in \mathbb{N}$, se tiene que $\forall x, y \in \mathbb{R}^n$:*

$$\langle x, y\rangle := \sum_{i=1}^{n} x_i y_i$$

*es un producto escalar, además:*

$$\|x\|_2 = \langle x, x\rangle^{1/2}$$

A partir de aquí, como es habitual, tomaremos el espacio vectorial $(\mathbb{R}^n, \langle\cdot,\cdot\rangle)$ el cual nos proporciona un producto escalar y una norma al mismo tiempo.

## 1.2. Conceptos básicos del Aprendizaje Automático

Como su nombre bien dice, el Aprendizaje Automático surge por la necesidad de conseguir que los ordenadores realicen ciertas tareas, de forma automática y a través de un método que les permita de manera autónoma ir corrigiendo sus propios errores (aprender) hasta obtener el resultado deseado. La gran diferencia entre estas nuevas técnicas y los algoritmos usuales es la de poder responder ante nuevas situaciones. Por ejemplo, ¿Cómo programamos un algoritmo que dada una nueva foto con un dígito entre el 0 y el 9 nos diga cuál de ellos es? La cantidad de fotos que hay es inmensa y ante una foto nunca vista el algoritmo no sabría qué responder, de ahí que surja la necesidad de encontrar un modelo que habiéndole mostrado una serie de dígitos, ante nuevos dígitos (situaciones nunca vistas) responda de forma correcta.

**Definición 1.3** (Conjunto de entrenamiento)**.** *Sea $g^* : X \to Y$ una aplicación suprayectiva*[1]*. Un conjunto de entrenamiento de la función $g^*$ es un conjunto finito $D \subset X \times Y$. A $X$ se le llama el conjunto de entradas de la función y a $Y$ el conjunto de salidas.*

En esencia, a la hora de aproximar cierta función $g^* : X \to Y$ se distinguen dos caminos a seguir como los más importantes dentro del aprendizaje automático: el *aprendizaje supervisado* y el *aprendizaje no supervisado.*

- Cuando el conjunto de entrenamiento sea de la forma: $\{x_t, g^*(x_t)\}_{t \leq T}$, $T \in \mathbb{N}$, estaremos hablando de aprendizaje supervisado. Es decir, proporcionamos información completa de nuestra función $g^*$ y a través de esos ejemplos debemos ser capaces de predecir de forma correcta $g^*(x)$ para cada $x \in X$. Como ejemplo podemos citar el ya mencionado reconocimiento de un número entre el 0 y el 9 en una foto con alguno de esos dígitos.

- Cuando el conjunto de entrenamiento sea de la forma: $\{x_t\}_{t \leq T} \times \emptyset$, estaremos hablando de aprendizaje no supervisado. Aquí no conocemos cuáles son las salidas de $g^*$, por lo que las estrategias, en este caso, consisten en encontrar cierta similitud entre nuestras entradas. Algunos ejemplos de aprendizaje no supervisado podrían ser: ordenar los libros de una biblioteca por su contenido (matemáticas, historia...), agrupar palabras con cierta relación entre sí (límite, derivada, integral...).

En este trabajo vamos a explicar un caso particular de red neuronal, cuyas técnicas están basadas en aprender ciertos ejemplos y responder de forma correcta ante nuevas situaciones y por tanto, forman parte del aprendizaje supervisado.

**Definición 1.4** (Conjunto de test)**.** *Sea $g^* : X \to Y$ una aplicación suprayectiva. Un conjunto de test de la función $g^*$ es un conjunto finito $D' \subset X \times Y$. Con $X$ las entradas e $Y$ las salidas.*

Los modelos de aprendizaje supervisado toman un conjunto de entrenamiento $D$ y otro de test $D'$ disjuntos. Son entrenados con $D$ para luego medir su precisión usando $D'$ y así comprobar que ante ejemplos nunca vistos devuelven una salida correcta.

**Definición 1.5** (Aproximación exacta)**.** *Sean $g^* : X \to Y$ una aplicación y $\{x_t, g^*(x_t)\}_{t \leq T}$ un conjunto de entrenamiento. Diremos que $f : X \to Y$ es una aproximación exacta de $g^*$ sobre ese conjunto de entrenamiento si se verifica:*

$$g^*(x_t) = f(x_t); \ \ \forall t \in \{1, ..., T\}$$

[1] $g^*(X) := \{g^*(x); x \in X\} = Y$.

## 1.3. Modelo matemático de neurona y limitaciones

A grandes rasgos, una neurona biológica recibe y transmite señales eléctricas (impulsos nerviosos) hacia otras células, por lo que matemáticamente, podemos pensar que una neurona recibe una serie de entradas $x \in \mathbb{R}^n$ para las cuales devuelve una salida $y = y(x) \in \mathbb{R}$.

**Definición 1.6** (Neurona)**.** *Una* ***neurona*** *es una tupla de cuatro elementos $w$, $b$, $S$ y $\sigma$ de manera que:*

1. *$w \in \mathbb{R}^n$ es su* ***vector de pesos****.*
2. *$b \in \mathbb{R}$ es su* ***umbral de activación*** *(Sirve de traslación).*
3. *$S : \mathbb{R}^n \to \mathbb{R}$ se le denomina* ***función de agregación****, a cada entrada $x \in \mathbb{R}^n$ le hace corresponder el valor $S(x) = \langle x, w\rangle - b$.*
4. *$\sigma : \mathbb{R} \to \mathbb{R}$ es una* ***función de activación*** *asociada a ella.*

La idea es la siguiente: la neurona recibe un vector de entradas $x \in \mathbb{R}^n$, hace la agregación $S(x) = \langle x, w\rangle - b$ y devuelve la salida correspondiente $\sigma(\langle x, w\rangle - b)$. Fijada una función de activación, la salida dependerá de los valores que hayamos tomado en los pesos $w$ y en su umbral de activación $b$.

Para empezar, ya se ve que una neurona no es suficiente para aproximar cualquier función de $\mathbb{R}^n$ a $\mathbb{R}^m$ ya que la salida de la misma va a parar a $\mathbb{R}$, aún así, una sola neurona no sirve para aproximar cualquier función de $\mathbb{R}^n$ en $\mathbb{R}$ ya que solamente disponemos de un vector de pesos y un umbral de activación para devolver la salida $\sigma(\langle x, w\rangle - b)$, la cual se queda corta y en muchas ocasiones no será suficiente.

Vamos a poner uno de los primeros ejemplos de neurona que surgieron, el *perceptrón simple*, cuyo fin era el de aproximar un tipo de funciones muy sencillas: *funciones lógicas*. De esta manera, podremos ver las limitaciones que tiene considerar una sola neurona.

**Definición 1.7** (Función lógica)**.** *Una función lógica es una aplicación $z : \{0,1\}^n \to \{0,1\}$ con tres operaciones posibles para las variables $(x_1, ..., x_n) \in \{0,1\}^n$:*

| $+$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

| $\cdot$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| $\bar{\cdot}$ | |
|---|---|
| 0 | 1 |
| 1 | 0 |

*La tabla de la suma se corresponde con la operación lógica $OR$, la tabla del producto con la operación $AND$ y la última tabla con la negación $NOT$.*

Lo primero que se preguntaron fue: ¿Podrá algún tipo de neurona clasificar correctamente las entradas y salidas de cierta función lógica? así, surgió el perceptrón simple.

**Ejemplo 1.3** (Perceptrón)**.** *Cuando la función de activación sea de la forma:*

$$\sigma(x) = \begin{cases} 1 & si \quad x \geq 0 \\ \\ 0 & si \quad x < 0 \end{cases} \tag{1.1}$$

*diremos que la neurona es un perceptrón, además, si restringimos las entradas de nuestra neurona a* $\{0,1\}^n$ *estaremos hablando del perceptrón simple. Por tanto, dada una función lógica* $z(x_1, ..., x_n)$*, debemos encontrar unos pesos* $(w_1, ..., w_n) \in \mathbb{R}^n$ *y un umbral* $b \in \mathbb{R}$ *de manera que se cumpla:*

$$\sigma(x_1 \cdot w_1 + ... + x_n \cdot w_n - b) = z(x_1, ..., x_n); \ \ \forall (x_1, ..., x_n) \in \{0,1\}^n$$

*pronto se dieron cuenta de que esto no era posible. Vamos a poner dos ejemplos en* $\{0,1\}^2$*:*

*Intentemos clasificar las entradas y salidas de la función lógica OR* $z_1(x_1, x_2) = x_1 + x_2$*. Se tiene que* $z_1(0,0) = 0$ *y* $z_1(0,1) = z_1(1,0) = z_1(1,1) = 1$*. Por lo que se debe cumplir:*

$$\sigma(-b) = 0, \ \ \sigma(w_1 - b) = 1, \ \ \sigma(w_2 - b) = 1 \ \ y \ \ \sigma(w_1 + w_2 - b) = 1$$

*de donde se deduce que:* $b > 0$*,* $w_1 \geq b$*,* $w_2 \geq b$ *y* $w_1 + w_2 \geq b$*. Podemos tomar* $b = 1$ *y* $w = (1,1)$ *y así:*

$$\sigma(x_1 + x_2 - 1) = \begin{cases} 1 & si \quad x_1 + x_2 - 1 \geq 0 \\ \\ 0 & si \quad x_1 + x_2 - 1 < 0 \end{cases}$$

*hemos conseguido encontrar un perceptrón simple que clasifica correctamente las entradas y salidas de la función lógica OR.*

*Intentemos clasificar ahora la función lógica* $z_2(x_1, x_2) = x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2$*, la cual recibe el nombre de XOR. Se tiene que* $z_2(0,0) = z_2(1,1) = 0$ *y* $z_2(1,0) = z_2(0,1) = 1$*. Esta vez se debe cumplir:*

$$\sigma(-b) = 0, \ \ \sigma(w_1 - b) = 1, \ \ \sigma(w_2 - b) = 1 \ \ y \ \ \sigma(w_1 + w_2 - b) = 0$$

*donde se deduce que:* $b > 0$*,* $w_1 \geq b$*,* $w_2 \geq b$ *y* $w_1 + w_2 < b$*, lo cual es imposible.*

Vemos que el primer intento de neurona no resultó del todo exitoso, por lo que se intentó encontrar una respuesta de cuándo un perceptrón iba a ser capaz de clasificar correctamente las entradas y salidas, esta vez no sólo de una función lógica, sino de forma más general cualquier aplicación binaria $g^* : X \to \{0,1\}$, $X \subset \mathbb{R}^n$.

**Definición 1.8** (Linealmente separables)**.** *Sean $A$ y $A'$ dos subconjuntos no vacíos de $\mathbb{R}^n$. Diremos que $A$ y $A'$ son linealmente separables si existen $w \in \mathbb{R}^n$ y $b \in \mathbb{R}$ verificando:*

1. $\langle a, w\rangle \geq b, \ \ \forall a \in A.$
2. $\langle a', w\rangle < b, \ \ \forall a' \in A'.$

*Geométricamente, $A$ y $A'$ son linealmente separables si podemos encontrar un hiperplano que corte el espacio $\mathbb{R}^n$ en dos, dejando a cada conjunto en uno de los dos lados.*

Notar que una condición necesaria para que dos conjuntos sean linealmente separables es que sean disjuntos, ya que si no fuese así, existiría $c \in A \cap A'$, lo que implicaría que: $\langle c, w\rangle < b \leq \langle c, w\rangle$.

**Figura 1.1:** En $\mathbb{R}^2$, dos conjuntos linealmente separables (izquierda) y dos conjuntos disjuntos que no son linealmente separables (derecha).

**Definición 1.9** (Aplicación linealmente separable)**.** *Sean $X \subset \mathbb{R}^n$ y $g^* : X \to \{0, 1\}$ una aplicación suprayectiva. Se dice que $g^*$ es una aplicación linealmente separable si existen $A, A'$ de manera que:*

1. *$A$ y $A'$ son linealmente separables.*
2. $A \cup A' = X.$
3. $g^*(A) \cap g^*(A') = \emptyset.$

En la Figura 1.1, nos podemos hacer la pregunta de sí un perceptrón podrá clasificar correctamente la aplicación:

$$g^* : \{Naranjas, Azules\} \to \{0, 1\}$$

pronto nos daremos cuenta de que solo es posible en el caso de la izquierda. Con las funciones lógicas pasaba algo parecido, si nos fijamos en la tabla de la operación $OR$ o $AND$, vemos que podemos separar por una linea recta los ceros y los unos, en cambio, para la función $XOR$ no es posible:

| $XOR$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

**Teorema 1.1** (Caracterización de los perceptrones)**.** *Sea $g^* : X \to \{0,1\}$ una aplicación suprayectiva, $X \subset \mathbb{R}^n$. Existe un perceptrón que clasifica correctamente las entradas y salidas de $g^*$ si y sólo si $g^*$ es una función linealmente separable.*

*Demostración.*

$\Rightarrow$

Consideramos el perceptrón $P_{w,b}$: $w \in \mathbb{R}^n$, $b \in \mathbb{R}$, $S(x) = \langle x, w\rangle - b$, $\forall x \in \mathbb{R}^n$ y la función habitual:

$$\sigma(\langle x, w\rangle - b) = \begin{cases} 1 \quad si \quad \langle x, w\rangle - b \geq 0 \\ \\ 0 \quad si \quad \langle x, w\rangle - b < 0 \end{cases}$$

que además verifica: $\sigma(\langle x, w\rangle - b) = g^*(x)$, $\forall x \in X$. Veamos que $g^*$ es una función linealmente separable:

Vamos a tomar los conjuntos $A = \{x \in X;\ g^*(x) = 0\}$ y $A' = \{x \in X;\ g^*(x) = 1\}$ y comprobar que se cumplen las propiedades 1, 2 y 3:

$1 \to A$ y $A'$ son linealmente separables, ya que por hipótesis, $\sigma(\langle x, w\rangle - b) = g^*(x)\ \forall x \in X$, lo que implica que, si $a \in A$ y $a' \in A'$:

$$\langle a, w\rangle < b \leq \langle a', w\rangle$$

$2 \to A \cup A' = X$.

$3 \to g^*(A) \cap g^*(A') = \{0\} \cap \{1\} = \emptyset$.

$\Leftarrow$

Supongamos que se cumplen las propiedades 1, 2 y 3 de función linealmente separable para $g^*$. Los conjuntos $A$ y $A'$ definidos como antes son los que nos las verifican, ya que si existieran otros $U, V \subset X$ distintos, se tendría que o bien $0, 1 \in g^*(U)$ o bien $0, 1 \in g^*(V)$. Como $g^*$ es suprayectiva y $g^*(U) \cap g^*(V) = \emptyset$, llegamos a que alguno de los dos deberá ser vacío y así (por ser $g^*$ suprayectiva) o bien $U$, o bien $V$ será vacío, contradiciéndose que sean linealmente separables (ya que ambos deben ser no vacíos).

Sabemos que los conjuntos $A$ y $A'$ son los responsables de que $g^*$ sea una función linealmente separable. Por ser ellos linealmente separables existirán $w \in \mathbb{R}^n$ y $b \in \mathbb{R}$ verificando $\forall a \in A$ y $\forall a' \in A'$:

$$\langle a, w\rangle < b \leq \langle a', w\rangle$$

por tanto, con ese $w \in \mathbb{R}^n$ y ese $b \in \mathbb{R}$ queda definido nuestro perceptrón.

□

El perceptrón fue de las primeras neuronas para las cuales se tenían fundamentos teóricos que respaldaban su funcionamiento. Podemos interpretarlo como un tipo de regresión lineal, que busca separar dos conjuntos con un hiperplano en $\mathbb{R}^n$. Esto es solo un ejemplo de neurona muy limitado por su función de activación (ya que devuelve 0 o 1) cuya finalidad es la de clasificar. Se va a necesitar la colaboración de varias neuronas para poder afrontar problemas de mayor complejidad. Surgen de esta forma las redes neuronales.

## 1.4. Arquitectura de una red neuronal hacia delante

Tenemos el concepto de neurona formalizado, simplemente consiste en recibir unas entradas y devolver unas salidas. Considerar una sola neurona se queda corto en muchas ocasiones, como hemos visto para el caso particular del perceptrón. Imaginemos ahora que queremos conectar varias de estas neuronas, de manera que las salidas de algunas de ellas sean entradas de otras. Dependiendo de cómo las agrupemos y conectemos obtenemos unos esquemas de neuronas distintos, los cuales formalmente se conocen como la *Arquitectura* de la red neuronal. Se representa mediante un *grafo* y es una forma simple y fácil de expresar nuestro modelo concreto de red neuronal, más potente como veremos que usar una sola neurona.

**Ejemplo 1.4** (Arquitectura de una neurona)**.** *Teniendo en cuenta el esquema que describimos de: llega un vector de entradas $x = (x_1, ..., x_n) \in \mathbb{R}^n$, se hace la agregación $S(x) = \langle w, x\rangle - b$ con los pesos $w = (w_1, ..., w_n) \in \mathbb{R}^n$ y su umbral $b \in \mathbb{R}$ y por último se devuelve la salida $\sigma(\langle w, x\rangle - b)$, donde $\sigma$ es la función de activación, podemos representar la siguiente arquitectura:*

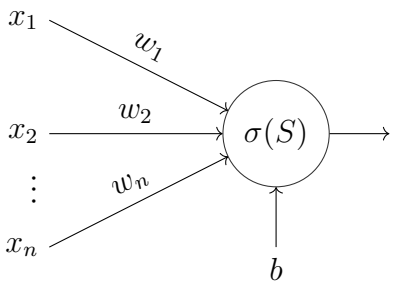


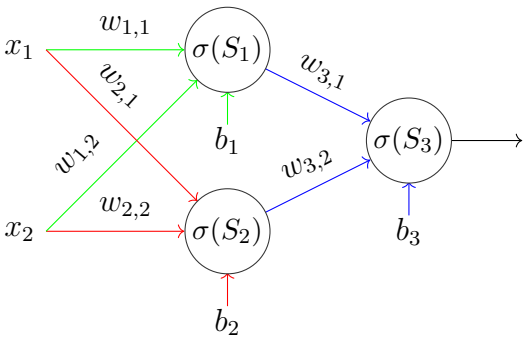

**Figura 1.2:** Arquitectura de una neurona.

Ahora, vamos a poner un ejemplo para ver que en efecto, conectando varias neuronas, somos capaces de resolver problemas más complejos.

**Ejemplo 1.5.** *Vamos a considerar tres perceptrones simples $P_{w_1,b_1}$, $P_{w_2,b_2}$ y $P_{w_3,b_3}$ con entradas en $\{0,1\}^2$, pesos $w_i = (w_{i,1}, w_{i,2}) \in \mathbb{R}^2$, umbrales $b_i \in \mathbb{R}$ y su función de activación $\sigma(S_i) = \sigma(\langle w_i, x\rangle - b_i)$, $i \in \{1,2,3\}$.*

$$\sigma(S_i) = \begin{cases} 1 & si \quad S_i \geq 0 \\ \\ 0 & si \quad S_i < 0 \end{cases}$$

*Los vamos a conectar atendiendo al siguiente esquema (Arquitectura):*




*notar que las dos salidas que nos proporcionan $P_{w_1,b_1}$ y $P_{w_2,b_2}$ forman el vector de entradas de $P_{w_3,b_3}$. Comprobemos si podemos representar con este esquema la función lógica XOR (recordar que en el ejemplo anterior no pudimos con un solo perceptrón simple).*

*Teníamos: $z(x_1,x_2) = x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2$. En este caso se debe verificar que $\sigma(S_3) = z(x_1,x_2)$, $\forall (x_1,x_2) \in \{0,1\}^2$.*

$$\sigma(S_3) = \sigma(w_{3,1} \cdot \sigma(w_{1,1} \cdot x_1 + w_{1,2} \cdot x_2 - b_1) + w_{3,2} \cdot \sigma(w_{2,1} \cdot x_1 + w_{2,2} \cdot x_2 - b_2) - b_3)$$

*considerando los siguientes valores: $b_1 = b_2 = b_3 = 1$, $w_1 = (-3,1)$, $w_2 = (1,-3)$ y $w_3 = (1,2)$ conseguimos modelar la función lógica $XOR$ mediante una red neuronal.*

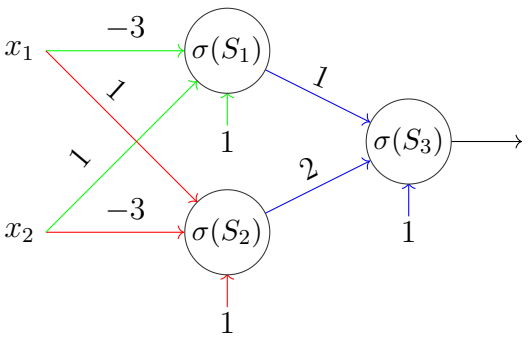



**Figura 1.3:** Arquitectura capaz de clasificar las entradas y salidas de la función lógica XOR.

Notar que hemos cogido la misma función de activación para todas las neuronas de la red, esto no tiene porque ser así. De hecho, este esquema es una de las muchas arquitecturas que existen, además, concuerda con la arquitectura de *red hacia delante* que queremos definir, la cual tiene las siguientes características:

- Las neuronas están agrupadas por *capas*.
- La función de activación es la misma para todas las neuronas, a excepción de la última capa (depende de las características de nuestro problema).
- Solo hay conexión entre capas vecinas, no se permite conexión entre neuronas de mismas capas o capas no vecinas.
- Las entradas de las neuronas de una capa son todas las salidas de la capa anterior (*Red totalmente conectada*).

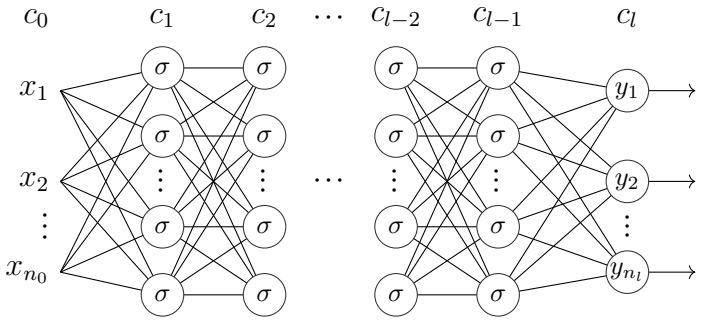



**Figura 1.4:** Arquitectura de una red neuronal hacia delante con $n_0$ entradas y $n_l$ salidas.

En la arquitectura de red neuronal incluida en la Figuara 1.4. cada $c_i$, $i \in \{1, ..., l\}$, denota la capa $i$-ésima, $l \in \mathbb{N}$ el número de capas, $x \in \mathbb{R}^{n_0}$ el vector de entradas e $y \in \mathbb{R}^{n_l}$ el vector de salidas.

A las capas $c_1, ..., c_{l-1}$ se le denominan *capas ocultas*, a la capa $c_0$ *capa de entrada* y a la capa $c_l$ *capa de salida*, además, si denotamos como $n_i :=$ número de neuronas en la capa $i-$ésima, podemos definir una notación bastante simplificada para referirnos a nuestra arquitectura: $(n_0, n_1, ..., n_{l-1}, n_l)$. Notar que si damos una función de activación $\sigma$ y unos pesos y umbrales concretos a cada neurona, nuestra red neuronal quedará totalmente definida[2].

En estas dos plataformas: [2, 3] se encuentran bastante bien explicadas otro tipo de arquitecturas y herramientas que pueden ser aplicadas para crear redes neuronales más complejas. Se dan referencias a distintos artículos donde se explican muchas de ellas.

[2]Estas dos componentes no dependen de la arquitectura de la red.

**Definición 1.10** (Matriz de pesos y umbrales)**.** *Sea $(n_0, ..., n_l)$ una arquitectura hacia delante. La matriz $W_j := (w^j_{i,k}|b^j_i)_{i\leq n_j, k\leq n_{j-1}}$ se define como la matriz de los pesos y los umbrales de las neuronas de la capa $c_j$, $j \in \{1, ..., l\}$.*

$$W_j = \left(\begin{array}{cccc|c} w^j_{1,1} & w^j_{1,2} & \cdots & w^j_{1,n_{j-1}} & b^j_1 \\ w^j_{2,1} & w^j_{2,2} & \cdots & w^j_{2,n_{j-1}} & b^j_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^j_{n_j,1} & w^j_{n_j,2} & \cdots & w^j_{n_j,n_{j-1}} & b^j_{n_j} \end{array}\right) \tag{1.2}$$

Notar que la fila $i-$ésima de la matriz se corresponde con los pesos y el umbral de la neurona $i-$ésima en la capa $c_j$. Para simplificarnos las cosas, vamos a definir la siguiente notación: dado $x = (x_1, ..., x_n) \in \mathbb{R}^n$ y $\sigma : \mathbb{R} \to \mathbb{R}$ una función:

$$\sigma\left(\begin{pmatrix} x_1 \\ x_2 \\ : \\ x_n \end{pmatrix}\right) := \begin{pmatrix} \sigma(x_1) \\ \sigma(x_2) \\ : \\ \sigma(x_n) \end{pmatrix}$$

Esta notación nos va a ayudar a definir de forma recursiva el tránsito de las entradas por nuestra red neuronal hasta las salidas.

**Definición 1.11** (Función de transición capas ocultas)**.** *Sea $(n_0, ..., n_l)$ una arquitectura hacia delante, sea $\sigma$ la función de activación de sus neuronas. La función de transición entre $c_{j-1}$ y $c_j$, $j \in \{1, ..., l-1\}$, se define como $\sigma_j$ y proporciona la salida de la capa $c_j$. Esto es, si $W_j$ es la matriz de pesos y umbrales de $c_j$, entonces:*

$$\begin{aligned} \sigma_j(W_j, \cdot) : \mathbb{R}^{n_{j-1}} &\to \mathbb{R}^{n_j} \\ x &\longmapsto \sigma_j(W_j, x) \end{aligned}$$

*Aquí:*

$$\sigma_j(W_j, x) = \sigma\left(\begin{pmatrix} w^j_{1,1} & \cdots & w^j_{1,n_{j-1}} \\ \vdots & \ddots & \vdots \\ w^j_{n_j,1} & \cdots & w^j_{n_j,n_{j-1}} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_{n_{j-1}} \end{pmatrix} - \begin{pmatrix} b^j_1 \\ \vdots \\ b^j_{n_j} \end{pmatrix}\right) \tag{1.3}$$

Notar que $\sigma_j(W_j, x)$ es un vector de $n_j$ componentes, donde la componente $i-$ésima representa la salida proporcionado por la neurona $i-$ésima en la capa $c_j$. Con $j \in \{1, ..., l-1\}$ e $i \in \{1, ..., n_j\}$.

**Definición 1.12** (Función de transición capa de salida)**.** *Sea $(n_0, ..., n_l)$ una arquitectura hacia delante, sean $y = (y_1, ..., y_{n_l}) \in \mathbb{R}^{n_l}$ las funciones de activación de la capa $c_l$. La función de transición entre $c_{l-1}$ y $c_l$ se define como $\sigma_l$ y proporciona la salida de la capa final $c_l$. Esto es, si $W_l$ es la matriz de pesos y umbrales de $c_l$, entonces:*

$$\begin{aligned} \sigma_l(W_l, \cdot) : \mathbb{R}^{n_{l-1}} &\to \mathbb{R}^{n_l} \\ x &\longmapsto \sigma_l(W_l, x) \end{aligned}$$

*Aquí:*

$$\sigma_l(W_l, x) = \begin{pmatrix} y_1\left(w^l_{1,1} \cdot x_1 + ... + w^l_{1,n_{l-1}} \cdot x_{n_{l-1}} - b^l_1\right) \\ \vdots \\ y_{n_l}\left(w^l_{n_l,1} \cdot x_1 + ... + w^l_{n_l,n_{l-1}} \cdot x_{n_{l-1}} - b^l_{n_l}\right) \end{pmatrix} \tag{1.4}$$

**Ejemplo 1.6.** *La red neuronal que modelaba la función lógica $XOR$ tiene una arquitectura $(2, 2, 1)$ y la misma función de activación (la función del perceptrón) tanto en la capa oculta como en la capa de salida.*

$$W_1 = \left(\begin{array}{cc|c} -3 & 1 & 1 \\ 1 & -3 & 1 \end{array}\right); \quad W_2 = \left(\begin{array}{cc|c} 1 & 2 & 1 \end{array}\right)$$

$$\sigma_1(W_1, (x, y)) = (\sigma(-3x + y - 1), \sigma(x - 3y - 1))$$

$$\sigma_2(W_2, (x, y)) = \sigma(x + 2y - 1)$$

Notar además que:

$$\sigma_2 \circ \sigma_1((x, y)) = \sigma\left(\sigma(-3x + y - 1) + 2\sigma(x - 3y - 1) - 1\right)$$

nos muestra la salida final de la red proporcionada por la entrada $(x, y) \in \mathbb{R}^2$. Esto motiva la siguiente definición:

**Definición 1.13** (Función salida)**.** *Sea $(n_0, ..., n_l)$ una arquitectura hacia delante. Vamos a definir el siguiente **vector de matrices** para referirnos a los pesos y umbrales totales de la red:*

$$W := (W_1, ..., W_l)$$

*donde cada $W_j$ se corresponde con la matriz de pesos y umbrales de la capa $c_j$, $j \in \{1, ..., l\}$. Sean $\{\sigma_j(W_j, \cdot)\}_{j \leq l}$ las funciones de transición entre capas, entonces la función salida de la red se define como:*

$$F(W,\cdot):\mathbb{R}^{n_0}\to\mathbb{R}^{n_l} \qquad (1.5)$$
$$x \longmapsto \sigma_l\circ\sigma_{l-1}\circ ...\circ\sigma_1(x)$$

Lo que hace esta función es comunicar todas las capas de nuestra red, transitándolas hasta devolver una salida. Vamos a definir el siguiente conjunto:

$$\mathcal{W} := M_{n_1\times n_0+1}(\mathbb{R})\times ...\times M_{n_l\times n_{l-1}+1}(\mathbb{R}) \qquad (1.6)$$

Notar que $W\in\mathcal{W}$ es un vector de matrices, por tanto, $\mathcal{W}$ es el conjunto de todos los posibles pesos y umbrales que podría tener una arquitectura $(n_0,...,n_l)$.

Dada una función $g^*:\mathbb{R}^{n_0}\to\mathbb{R}^{n_l}$, el objetivo será encontrar $W^*\in\mathcal{W}$ para que $F(W^*,\cdot)$ se asemeje lo máximo posible a $g^*$. Bajo ciertas hipótesis, como la naturaleza de la función $g^*$, el entorno donde queramos aproximarla o las características que tenga nuestra función de activación, se puede garantizar una correcta aproximación. Por ejemplo, si utilizamos perceptrones en la salida de nuestra arquitectura es imposible aproximar cualquier función, ya que las salidas de $F(W^*,\cdot)$ caerán en $\{0,1\}^{n_l}$ (ya que $\sigma(x)$ era o 0 o 1).

Incluso suponiendo que podemos garantizar la existencia de un $W^*\in\mathcal{W}$ el cual nos proporciona una buena aproximación de $g^*$ ¿Cómo lo calculamos? En los ejemplos que hemos puesto no hemos tenido dificultades, no obstante, en la mayoría de las ocasiones no va a ser así, por lo que necesitaremos un método que nos permita calcularlo, o al menos aproximarlo.

Ahora, debemos ver si el modelo que hemos construido, en efecto, sirve para aproximar funciones de $\mathbb{R}^{n_0}$ en $\mathbb{R}^{n_l}$ y si es así, debemos encontrar un método que nos permita encontrar unos pesos y umbrales adecuados. Antes de pasar a el siguiente capítulo, pondremos una última sección con algunas de las funciones de activación más usadas.

## 1.5. Ejemplos de funciones de activación

Debemos ser capaces de encontrar una función de activación acorde a las características de nuestro problema, de hecho, la elección de una correcta función de activación es quizá la parte más importante antes de intentar aproximar, aquí ponemos algunos ejemplos:

$$\sigma(x)=\begin{cases}1 & si \quad x\geq 0\\ \\ 0 & si \quad x<0\end{cases} \qquad (1.7)$$

Esta función ya la habíamos mencionado antes, ya que es la función de activación del perceptrón. Una de sus utilidades, como ya hemos visto podrían ser la de clasificar de for-

ma binaria dos conjuntos. Posee una discontinuidad, no obstante, disponemos de algunas aproximaciones suaves como la *función logística:*

$$\sigma(x) = \frac{1}{1+e^{-x}}; \qquad \sigma'(x) = \sigma(x) \cdot (1 - \sigma(x)) \tag{1.8}$$

la cual forma parte de un grupo de funciones llamadas *funciones sigmoides.*

**Definición 1.14** (Función sigmoide)**.** *Una aplicación $\sigma : \mathbb{R} \to \mathbb{R}$ se dice sigmoide si satisface:*

$$\lim_{x \to -\infty} \sigma(x) = 0; \qquad \lim_{x \to \infty} \sigma(x) = 1$$

Es fácil encontrar funciones sigmoides. Por ejemplo, en el campo de las probabilidades, si $X$ es una variable aleatoria continua, entonces su función de distribución $F(x) = P(X \leq x)$ es una función sigmoide. Algunos ejemplos de funciones con características similares podrían ser:

$$\sigma(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \qquad \sigma'(x) = 1 - \sigma(x)^2 \tag{1.9}$$

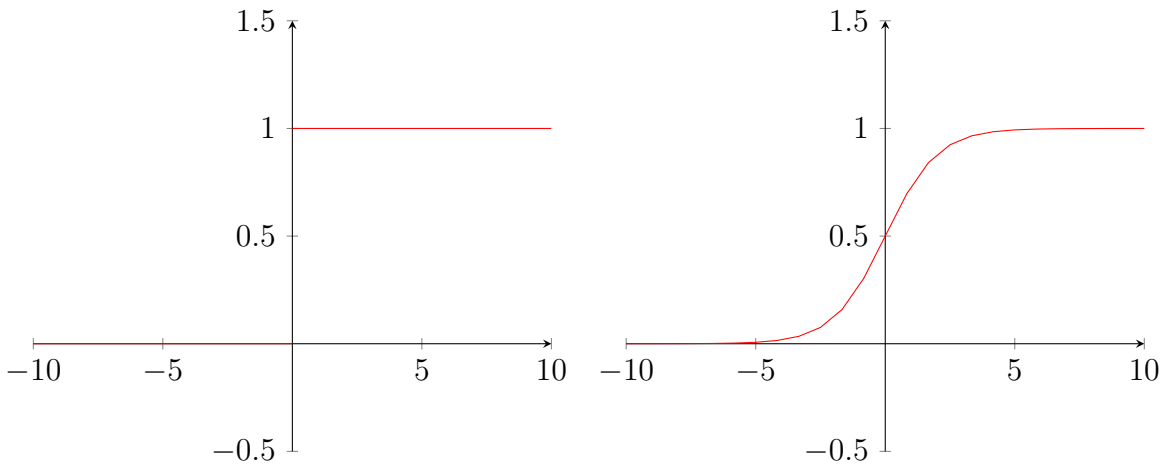



**Figura 1.5:** Gráfica de la función de activación del perceptrón simple. A la derecha se tiene una aproximación suya de clase $\mathcal{C}^\infty$ (la función logística).

Cabe resaltar que estas funciones de activación se suelen encontrar tanto en las capas ocultas como en la salida. Para problemas de clasificación múltiple existe otra función llamada *softmax*, se utiliza exclusivamente en la salida de nuestra red y funciona de la siguiente forma:

$$\begin{aligned} \Sigma\colon \mathbb{R}^{n_l} &\to \mathbb{R}^{n_l} \\ x &\longmapsto \Sigma(x) \end{aligned} \tag{1.10}$$

Aquí, tomando $e^x = (e^{x_1}, ..., e^{x_{n_l}})$:

$$\Sigma(x) = \frac{e^x}{\sum_{k=1}^{n_l} e^{x_k}} \in \mathbb{R}^{n_l}$$

Notar que esta función de activación se puede ver como $n_l$ funciones de activación distintas para nuestra salida $y := \sigma_l \in \mathbb{R}^{n_l}$, donde:

$$y_j = \frac{e^{x_j}}{\sum_{k=1}^{n_l} e^{x_k}} \in \mathbb{R}$$

en este caso, $n_l$ indica el número de clases distintas que tenemos. A cada clase se le asigna uno de los vectores canónicos $e_j \in \mathbb{R}^{n_l}$:

$$\begin{aligned} \text{Clase } 1 &\longrightarrow e_1 \\ &\vdots \\ \text{Clase } n_l &\longrightarrow e_{n_l} \end{aligned}$$

además, se tiene que:

$$\sum_{j=1}^{n_l} y_j = 1$$

por lo que esta función de activación se puede ver como una distribución de probabilidad, donde la componente $k-$ésima $y_k$ indica la probabilidad de que cierta entrada de nuestro modelo pertenezca a la Clase $k$.

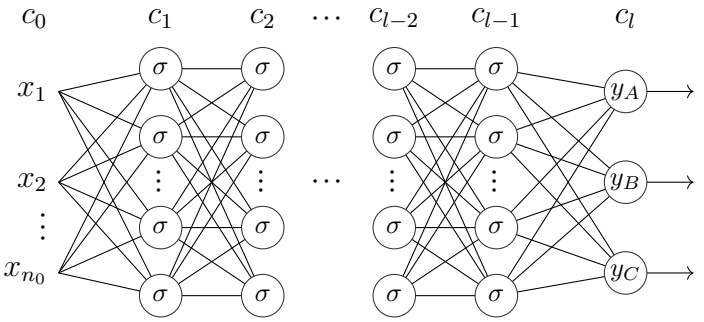
**Ejemplo 1.7** (Función softmax para 3 clases)**.** *Supongamos que estamos ante un problema de clasificación de tres clases A, B y C. Entonces* $n_l = 3$, $y = (y_A, y_B, y_C)$*:*

$$A \longrightarrow (1,0,0); \quad y_A = \frac{e^{x_A}}{e^{x_A} + e^{x_B} + e^{x_C}}$$

$$B \longrightarrow (0,1,0); \quad y_B = \frac{e^{x_B}}{e^{x_A} + e^{x_B} + e^{x_C}}$$

$$C \longrightarrow (0,0,1); \quad y_C = \frac{e^{x_C}}{e^{x_A} + e^{x_B} + e^{x_C}}$$

*Supongamos que tenemos una arquitectura arbitraria* $(n_0, ..., 3)$ *con una función de activación* $\sigma$*:*




*y que dada una entrada $x \in \mathbb{R}^{n_0}$ nuestra red neuronal devuelve una salida:*

$$F(W, x) = (0.01, 0.002, 0.988)$$

*entonces, según nuestro modelo, lo más probable es que esa entrada $x$ pertenezca a la clase $C$.*

Existen otro tipo de funciones de activación, entre ellas, de las más usadas en las capas ocultas es:

$$\sigma(x) = máx\{0, x\} \tag{1.11}$$

la cual recibe el nombre de *unidad de rectificador lineal* (ReLU). También tiene aproximaciones suyas de clase $\mathcal{C}^\infty$:

$$\sigma(x) = \ln(1 + e^x); \quad \sigma'(x) = \frac{1}{1 + e^{-x}} \tag{1.12}$$

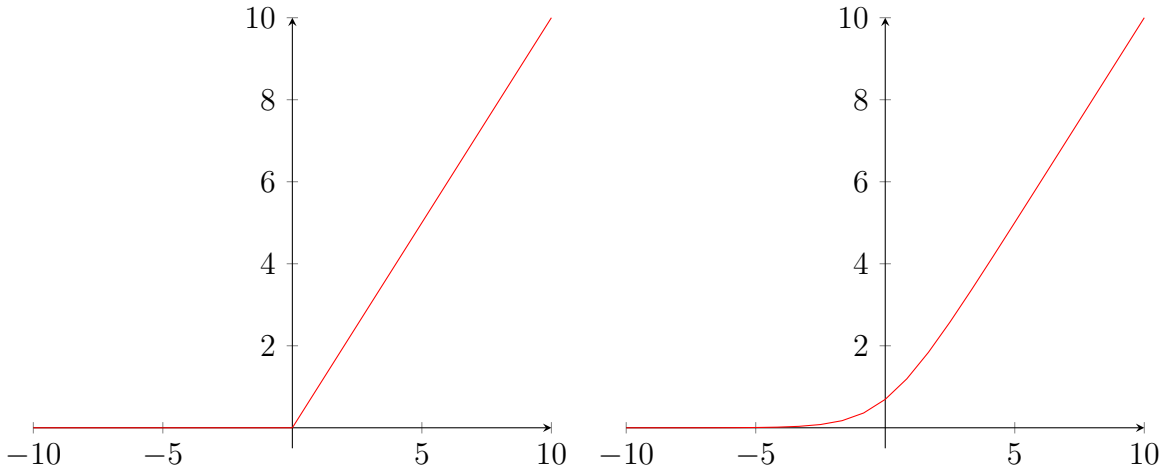



**Figura 1.6:** Gráfica de la función de activación ReLU (unidad de rectificación lineal), a la derecha se tiene una aproximación suya de clase $\mathcal{C}^\infty$.

# Capítulo 2

# Teorema de aproximación universal

Una vez hemos obtenido un modelo de red neuronal artificial, nos preguntamos si, en efecto, este modelo es útil y podemos aproximarnos a funciones. Los resultados que pondremos no son lo más recientes que hay, de hecho este problema sigue abierto y todavía hay muchas cuestiones por resolver como por ejemplo cuál es el número óptimo de capas y de neuronas en cada capa para la aproximación. Daremos un resultado muy concreto que nos garantiza que cualquier función $g^* : [0,1]^{n_0} \to \mathbb{R}^{n_2}$, bajo ciertas condiciones, se pude aproximar mediante una arquitectura con una sola capa oculta.

## 2.1. Fundamentos del análisis funcional

En esta sección simplemente nos vamos a parar brevemente a dar una serie de definiciones y enunciar algunos teoremas que nos servirán posteriormente para enfrentar nuestro problema, por lo que algunos de los resultados carecen de demostración.

Sea $I^n := [0,1]^n$. Vamos a considerar como nuestro espacio vectorial:

$$L^2(I^n) := \{f : I^n \to \mathbb{R};\ \int_{I^n} |f(x)|^2 dx < +\infty\}$$

a el cual le podemos asignar una norma[1], si $f \in L^2(I^n)$:

$$\|f\| := \left(\int_{I^n} |f(x)|^2 \, dx\right)^{1/2}$$

y un producto escalar, si $f, g \in L^2(I^n)$:

$$\langle f, g\rangle := \int_{I^n} f(x)g(x)dx$$

que verifican la relación:

[1]En realidad sería una semi-norma ya que $\|f\| = 0 \Rightarrow f$ es cero salvo en un conjunto de medida nula, no obstante podemos definir la relación de equivalencia $f \sim g \ \ sii \ \ \|f - g\| = 0$.

$$\|f\| = \langle f, f\rangle^{1/2},\ \forall f \in L^2(I^n)$$

**Definición 2.1** (Topología)**.** *Sea $X$ es un conjunto. Una topología en $X$ es una colección de subconjuntos suyos denotada como $\mathcal{T}$ verificando estas tres propiedades:*

1. $\emptyset, X \in \mathcal{T}$
2. $\{A_\lambda\}_{\lambda\in\Lambda} \subset \mathcal{T} \Rightarrow \cup_{\lambda\in\Lambda} A_\lambda \in \mathcal{T}$
3. $A, A' \in \mathcal{T} \Rightarrow A \cap A' \in \mathcal{T}$

*El par ($X$,$\mathcal{T}$) se dice que es un espacio topológico. Si no hay confusión sobre la topología, se suele denotar simplemente como $X$.*

**Definición 2.2** (Abiertos y cerrados)**.** *Sea $X$ un conjunto y $\mathcal{T}$ una topología en $X$. Sea $A \subset X$:*

1. *Si $A \in \mathcal{T}$ diremos que $A$ es **abierto**.*
2. *Si $A^c \in \mathcal{T}$ diremos que $A$ es **cerrado**.*

Un conjunto es abierto (cerrado) si y solo si su complementario es cerrado (abierto).

Dada $f \in L^2(I^n)$ y $r > 0$ definimos el siguiente conjunto:

$$B_r(f) := \{g \in L^2(I^n);\ \|f - g\| < r\} \subset L^2(I^n)$$

con el cual podemos definir la siguiente topología:

$$\mathcal{T} := \{A \subset L^2(I^n);\ \forall f \in A \text{ existe } r > 0 \text{ tal que } B_r(f) \subset A\}$$

de hecho, este es un caso particular de lo que llamamos una topología inducida por una métrica[2]. Por lo que nuestro espacio $L^2(I^n)$ es un espacio vectorial con un producto escalar y una norma, la cual a su vez nos proporciona una topología.

**Definición 2.3** (Clausura)**.** *Sean $X$ es un espacio topológico y $A \subset X$. Se define la clausura de $A$ en $X$ como:*

$$\overline{A} := \cap\{C;\ A \subset C\ y\ C\ cerrado\}$$

*esto es, $\overline{A}$ es el menor cerrado que contiene a $A$.*

[2]En este caso la métrica es la norma de $L^2(I^n)$.

**Definición 2.4** (Conjunto denso)**.** *Sea $(\mathcal{V}, \|\cdot\|)$ un espacio vectorial normado sobre $\mathbb{R}$. Diremos que $\mathcal{D} \subset \mathcal{V}$ es denso en $\mathcal{V}$ si para todo $x \in \mathcal{V}$ y $\epsilon > 0$ existe un $d \in \mathcal{D}$ verificando:*

$$\|x - d\| < \epsilon$$

En realidad, la definición más amplia de conjunto denso sobre cualquier espacio topológico es que su clausura sea el espacio total (en nuestro caso $\overline{\mathcal{D}} = \mathcal{V}$). No obstante, cuando se trata de un espacio métrico, esta definición y la que hemos dado son equivalentes. Si nos fijamos un poco en la definición, vemos que $\mathcal{D}$ es denso si para todo $x \in \mathcal{V}$ podemos encontrar un elemento en $\mathcal{D}$ tan cerca de él como nosotros queramos.

A partir de aquí no vamos a entrar en detalle de más definiciones, ya que solo son mencionadas en el teorema que viene a continuación, que no vamos a demostrar, y en el corolario que sigue. En concreto estamos hablando de la noción de aplicación continua, aplicación lineal, conjunto convexo y conjunto compacto.

**Teorema 2.1** (Teorema de Hahn-Banach)**.** *Sea $(\mathcal{V}, \|\cdot\|)$ un espacio vectorial normado sobre $\mathbb{R}$. Sean $A, A' \subset \mathcal{V}$ cerrados, no vacíos, disjuntos, convexos y al menos uno de los dos compacto. Existe una aplicación lineal y continua $T : \mathcal{V} \to \mathbb{R}$ no nula y $a \in \mathbb{R}$ tal que:*

1. $T(x) > a; \quad \forall x \in A$
2. $T(x) < a; \quad \forall x \in A'$

Notar que este teorema generaliza la noción de conjuntos linealmente separables que definimos en el capítulo anterior, además, podemos suponer que $a$ es positivo ya que la aplicación anterior $T$ no es única pues $-T$ también es una aplicación lineal y continua verificando la tesis de nuestro teorema. Si $a \geq 0$ utilizamos $T$ y si $a < 0$ podemos utilizar $-T$. La demostración se puede encontrar en [4].

**Corolario 2.2.** *Sean $(\mathcal{V}, \|\cdot\|)$ un espacio vectorial normado sobre $\mathbb{R}$ y $\mathcal{U} \subset \mathcal{V}$ un subespacio no denso ($\overline{\mathcal{U}} \neq \mathcal{V}$). Entonces existe una aplicación lineal, continua y no nula $T : \mathcal{V} \to \mathbb{R}$ tal que $T(x) = 0; \ \forall x \in \mathcal{U}$.*

*Demostración.* Sea $z \in \mathcal{V} \setminus \overline{\mathcal{U}}$. Notar que $\{z\}$ es cerrado, compacto y convexo:

Es cerrado por ser un unipuntual en un espacio métrico, es compacto por ser finito y es convexo ya que si $\lambda \in [0, 1]$, $z \cdot \lambda + (1 - \lambda) \cdot z = z \in \{z\}$.

Por otro lado tenemos que $\overline{\mathcal{U}}$ es cerrado por la propia definición de clausura y, además, por ser subespacio de $\mathcal{V}$, es convexo. Por último, notar que $\{z\} \cap \overline{\mathcal{U}} = \emptyset$.

Aplicando el Teorema de Hahn-Banach existirá una aplicación lineal, continua y no nula $T : \mathcal{V} \to \mathbb{R}$ y $a > 0$ verificando:

$$T(x) < a < T(z);\ \forall x \in \overline{\mathcal{U}}$$

Nos falta ver que es cero dentro de $\mathcal{U}$. Como $T$ es lineal, dado $n \in \mathbb{N}$ y $x \in \overline{\mathcal{U}}$:

$$T(n \cdot x) = n \cdot T(x) < a \Rightarrow T(x) < \frac{a}{n}$$

$$T(-n \cdot x) = -n \cdot T(x) < a \Rightarrow T(x) > -\frac{a}{n}$$

como $n$ es un número natural arbitrario llegamos a que $T(x) = 0;\ \forall x \in \overline{\mathcal{U}}$ y por tanto también será nula en $\mathcal{U}$.

□

Notar que hemos probado más, ya que hemos demostrado que se cumple no solo en $\mathcal{U}$, sino también en su clausura. También hemos dado por sabido en esta demostración que $\overline{\mathcal{U}}$ sigue siendo un subespacio de $\mathcal{V}$.

**Teorema 2.3** (Teorema de representación de Riesz)**.** *Sea $T : L^2(I^n) \to \mathbb{R}$ una aplicación lineal y continua, entonces existe una única $g \in L^2(I^n)$ tal que:*

$$T(f) = \langle f, g \rangle = \int_{I^n} f(x) \cdot g(x) dx,\ \forall f \in L^2(I^n)$$

Aunque es posible enunciar este teorema de una forma más general, la forma simple anterior es suficiente para lo que sigue en las siguientes secciones (ver [1] para la demostración).

**Definición 2.5.** *(Transformada de Fourier) Si $f \in L^2(I^n)$ se define la transformada de Fourier de $f$ como la función:*

$$\hat{f}(w) = \int_{I^n} e^{i\langle x,w \rangle} \cdot g(x) dx$$

## 2.2. Resultados preliminares

Vamos a considerar la siguiente arquitectura:

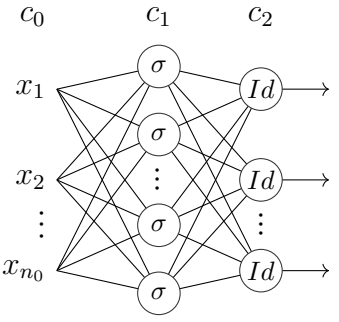


**Figura 2.1:** Arquitectura con una sola capa oculta $(n_0, n_1, n_2)$.

la cual tiene como funciones de activación $\sigma$ en $c_1$ y la identidad en $c_2$. Vamos a tomar las dos matrices de pesos de la siguiente manera:

$$W_1 = \left(\begin{array}{cccc|c} w^1_{1,1} & w^1_{1,2} & \cdots & w^1_{1,n_0} & b^1_1 \\ w^1_{2,1} & w^1_{2,2} & \cdots & w^1_{2,n_0} & b^j_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^1_{n_1,1} & w^1_{n_1,2} & \cdots & w^1_{n_1,n_0} & b^1_{n_1} \end{array}\right) \quad W_2 = \left(\begin{array}{cccc|c} w^2_{1,1} & w^2_{1,2} & \cdots & w^2_{1,n_1} & 0 \\ w^2_{2,1} & w^2_{2,2} & \cdots & w^2_{2,n_1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{n_2,1} & w^2_{n_2,2} & \cdots & w^2_{n_2,n_1} & 0 \end{array}\right) \tag{2.1}$$

entonces, la salida de la red neuronal a $\mathbb{R}^{n_2}$ tendrá la siguiente forma:

$$\begin{pmatrix} w^2_{1,1} & \cdots & w^2_{1,n_1} \\ \vdots & \ddots & \vdots \\ w^2_{n_2,1} & \cdots & w^2_{n_2,n_1} \end{pmatrix} \cdot \sigma \left( \begin{pmatrix} w^1_{1,1} & \cdots & w^1_{1,n_0} \\ \vdots & \ddots & \vdots \\ w^1_{n_1,1} & \cdots & w^1_{n_1,n_0} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_{n_0} \end{pmatrix} - \begin{pmatrix} b^1_1 \\ \vdots \\ b^1_{n_1} \end{pmatrix} \right) \tag{2.2}$$

Notar que la componente $i-$ésima de la salida es una *combinación lineal* de la forma:

$$\sum_{k=1}^{n_1} w^2_{i,k} \cdot \sigma(w^1_{k,1} \cdot x_1 + ... + w^1_{k,n_0} \cdot x_{n_0} - b^1_k) \tag{2.3}$$

Otra cosa interesante a tener en cuenta es que si queremos aproximarnos a cierta función $g^* : I^{n_0} \to \mathbb{R}^{n_2}$, se tiene:

$$g^*(x) = \left(g^*_1(x), ..., g^*_{n_2}(x)\right)$$

por tanto, podemos aproximarnos a $g^*$ componente a componente de manera independiente, utilizando combinaciones lineales como en 2.3, por lo que demostrando la aproximación para funciones $I^{n_0} \to \mathbb{R}$ obtenemos directamente también el resultado en $\mathbb{R}^{n_2}$. Vamos a mostrar esta idea para funciones de $I^2$ en $\mathbb{R}^2$:

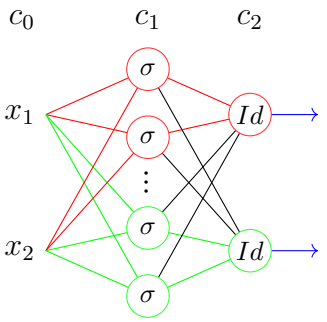
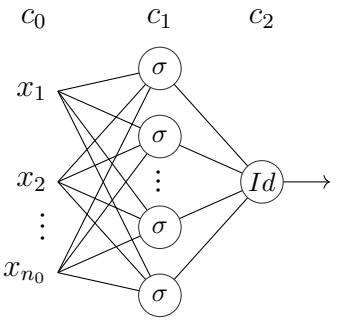



**Figura 2.2:** Ilustración de cómo una arquitectura $(2, n_1, 2)$ aproxima componente a componente cierta función $g^* : \mathbb{R}^2 \to \mathbb{R}^2$. Las lineas negras indican que los pesos en esa conexión se toma igual a cero.

Independientemente de que esta sea o no la mejor forma de aproximar una función, nos sirve de gran ayuda para lo que queremos probar, ya que ahora solo nos debemos preocupar de encontrar un número de capas ocultas $n_1$ y una función de activación $\sigma$ adecuados para ver cuándo somos capaces de acercarnos a funciones en $L^2(I^{n_0})$. Ahora nuestra situación es la siguiente:




$$W_1 = \left(\begin{array}{cccc|c} w^1_{1,1} & w^1_{1,2} & \cdots & w^1_{1,n_0} & b^1_1 \\ w^1_{2,1} & w^1_{2,2} & \cdots & w^1_{2,n_0} & b^j_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^1_{n_1,1} & w^1_{n_1,2} & \cdots & w^1_{n_1,n_0} & b^1_{n_1} \end{array}\right) \quad W_2 = \left(\begin{array}{cccc|c} w^2_{1,1} & w^2_{1,2} & \cdots & w^2_{1,n_1} & 0 \end{array}\right) \tag{2.4}$$

Definimos el siguiente conjunto (y **subespacio** de $L^2(I^{n_0})$):

$$\Sigma_{n_0}(\sigma) \;:=\; gen\{\sigma\left(\langle x, w\rangle - b\right);\;\; w \in \mathbb{R}^{n_0},\; b \in \mathbb{R}\} \tag{2.5}$$

que no es otra cosa que el conjunto de todas las posibles funciones de salida que toma una red neuronal con una sola capa oculta, $n_0$ entradas y 1 salida (el número de neuronas en la capa oculta $n_1$ se mantiene indeterminado) con función de activación $\sigma$ y salida la identidad. Por ejemplo, si el número de entradas $n_0 = 2$ :

$$2 \cdot \sigma(x_1 + 2 \cdot x_2 - 20) - 3 \cdot \sigma(-x_2 + 1) \in \Sigma_2(\sigma)$$
$$3 \cdot \sigma(20 \cdot x_1 - x_2 + 10^5) - 3 \cdot \sigma(-x_2 + 1) + \pi \cdot \sigma(x_1 + x_2) + 4 \in \Sigma_2(\sigma)$$

en el primer caso estaríamos mostrando una posible salida de una arquitectura $(2, 2, 1)$ y en el segundo la de una arquitectura $(2, 4, 1)$. Ahora, lo único que necesitamos saber es qué características debe tener nuestra función de activación $\sigma$ para que nuestro conjunto $\Sigma_{n_0}(\sigma)$ sea denso en $L^2(I^{n_0})$.

**Ejemplo 2.1** (Los polinomios no sirven)**.** *Sea $p_n : I^{n_0} \to \mathbb{R}$ un polinomio de grado $n$. Tenemos que:*

$$\Sigma_{n_0}(p_n) \subset \{\textit{Polinomios de grado} \leq n\}$$

*ya que al sumar combinaciones lineales de $p_n$ nos queda otro polinomio como mucho de grado $n$. El conjunto de polinomios de grado no mayor que $n$ no es denso en $L^2(I^{n_0})$ y por tanto $\Sigma_{n_0}(p_n)$ tampoco. Por ejemplo, para $n_0 = 1$ y tomando el polinomio identidad $x$ es fácil ver que:*

$$\Sigma_1(x) = \{Polinomios\ de\ grado \leq 1\}$$

*Se puede demostrar que para la función $x^3 \in L^2(I)$ su mejor aproximación en $\Sigma_1(x)$ es:*

$$p(x) = \frac{9}{10}x - \frac{1}{5} \in \Sigma_1(x)$$

$$\|x^3 - p(x)\| = \left(\int_{I^n} \left|x^3 - \frac{9}{10}x + \frac{1}{5}\right|^2 dx\right)^{1/2} = \frac{3}{10\sqrt{7}} \approx 0.1134$$

*por tanto, tomando $\epsilon = 0.1$ y $x^3 \in L^2(I)$ no existe $q(x) \in \Sigma_1(x)$ tal que $\|x^3 - q(x)\| < \epsilon$.*

**Definición 2.6** (Aproximación universal)**.** *Diremos que el conjunto de redes neuronales con una capa oculta, función de activación $\sigma$ y salida la identidad es una aproximación universal si $\Sigma_{n_0}(\sigma)$ es denso en $L^2(I^{n_0})$, $\forall n_0 \in \mathbb{N}$.*

**Definición 2.7.** *Sea $n_0 \in \mathbb{N}$. Diremos que una función $\sigma : \mathbb{R} \to \mathbb{R}$ es $n_0-$discriminatoria si la única función $g \in L^2(I^{n_0})$ que verifica:*

$$\int_{I^{n_0}} \sigma(\langle x, w\rangle - b) \cdot g(x)dx = 0;\ \ \forall w \in \mathbb{R}^{n_0}\ y\ \forall b \in \mathbb{R} \tag{2.6}$$

*es la función nula $g \equiv 0$ ($g = 0$ casi todo punto). Si esta propiedad 2.6 se cumple para todo $n_0 \in \mathbb{N}$ diremos que la función $\sigma$ es **discriminatoria**.*

## 2.3. Enunciado del teorema

**Teorema 2.4** (Teorema de aproximación universal)**.** *Sea $\sigma$ una función discriminatoria. Las redes neuronales con una capa oculta, función de activación $\sigma$ y salida la identidad son una aproximación universal.*

*Demostración.* Sea $n_0 \in \mathbb{N}$, debemos ver que el conjunto $\Sigma_{n_0}(\sigma)$ es denso en $L^2(I^{n_0})$, para ello supongamos que $\overline{\Sigma_{n_0}(f)} \neq L^2(I^{n_0})$ y busquemos una contradicción.

En la sección anterior probamos el Corolario 2.2. del Teorema de Hahn-Banach, que nos garantizaba que si $\Sigma_{n_0}(\sigma)$ no era denso en $L^2(I^{n_0})$, podemos encontrar una aplicación lineal, continua y **no nula** $T : L^2(I^{n_0}) \to \mathbb{R}$ tal que:

$$T(f) = 0;\ \forall f \in \Sigma_{n_0}(\sigma)$$

por otro lado, por el Teorema de representación de Riesz, tenemos que existirá una única función $g \in L^2(I^{n_0})$ verificando:

$$T(f) = \int_{I^{n_0}} f(x) \cdot g(x)dx;\ \forall f \in L^2(I^{n_0})$$

sabemos que para todo $w \in \mathbb{R}^{n_0}$ y $b \in \mathbb{R}$, $\sigma(\langle x, w\rangle - b) \in \Sigma_{n_0}(\sigma)$, además:

$$T\left(\sigma(\langle x, w\rangle - b)\right) = \int_{I^{n_0}} \sigma(\langle x, w\rangle - b) \cdot g(x)dx = 0$$

pero $\sigma$ es discriminatoria y por tanto $g \equiv 0 \Rightarrow T = 0$, lo cual contradice la afirmación anterior de que $T$ sea no nula.

□

**Ejemplo 2.2.** *La función del perceptrón es discriminatoria.*

$$\int_{I^n} \sigma(\langle x, w\rangle - b) \cdot g(x)dx = 0;\ \forall w \in \mathbb{R}^n\ y\ \forall b \in \mathbb{R}$$

*debemos ver que $g \equiv 0$, se tiene que:*

$$\sigma(\langle x, w\rangle - b) = \begin{cases} 1 & si \quad \langle x, w\rangle - b \geq 0 \\ \\ 0 & si \quad \langle x, w\rangle - b < 0 \end{cases}$$

*Definimos el siguiente conjunto: $\Omega_{w,b} := \{x \in I^n;\ \langle x, w\rangle - b \geq 0\}$.*

$$I_{w,b} := \int_{I^n} \sigma(\langle x, w\rangle - b) \cdot g(x)dx = \int_{\Omega_{w,b}} g(x)dx = 0;\ \forall w \in \mathbb{R}^n\ y\ \forall b \in \mathbb{R}$$

*la integral respecto a cualquier semiplano contenido en $I^n$ es cero. Si $g$ fuese positiva se probaría muy fácil ya que:*

$$\int_{I^n} g(x)dx = \int_{\Omega_{w,b}} g(x)dx + \int_{\Omega_{-w,-b}} g(x)dx = 0$$

*lo que implicaría que $g \equiv 0$ pero la función $g$ no necesariamente es positiva, por lo que hay que utilizar otra estrategia. Dada $h$ una función medible y acotada definimos la siguiente aplicación lineal:*

$$F(h) = \int_{I^n} h(\langle x, w\rangle - b) \cdot g(x)dx$$

*por hipótesis, tenemos que $F(\sigma) = 0$. La función $\sigma$ se puede ver como la función indicadora de el intervalo $[0, +\infty)$. Utilizando la linealidad de $F$ se puede ver que la función vale cero para cualquier función indicadora de un intervalo.*

$$F(f) = 0; \quad f(x) = \begin{cases} 1 & si \quad x \in [a, b] \\ \\ 0 & si \quad x \notin [a, b] \end{cases}$$

*Ahora, volviendo a utilizar la linealidad, llegamos a que $F(s) = 0$ si $s$ es una función simple (una combinación lineal de indicadoras). Utilizamos que las funciones simples son densas en el conjunto de las funciones medibles y acotadas y llegamos a que:*

$$F(h) = 0; \ \forall h \in L^{\infty}(\mathbb{R})$$

*Ahora tomamos $h(x) = e^{ix} \in L^{\infty}(\mathbb{R})$ y se tendrá:*

$$F(h) = e^{-ib} \int_{I^n} e^{i\langle x,w\rangle} \cdot g(x)dx = 0$$

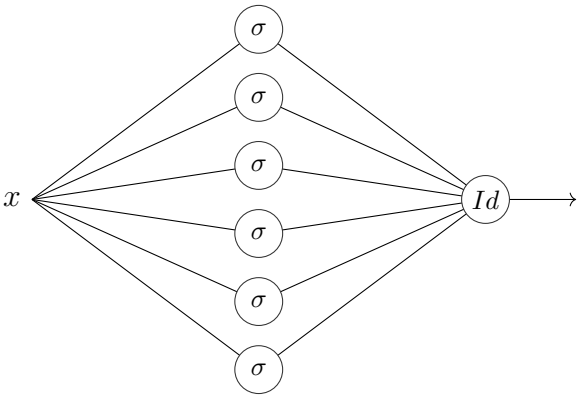
*llegamos a que la transformada de Fourier de la función $g$ es igual a cero, lo que implica que $g$ es igual a cero casi todo punto. Por tanto, $\sigma$ es discriminatoria.*

□

Notar que una vez demostrada la aproximación para $I^n$, automáticamente obtenemos la prueba para cualquier rectángulo en $\mathbb{R}^n$. Es de esperar que ciertas funciones de activación en la salida distintas de la identidad sirvan también para aproximar cualquier función, como por ejemplo $\sigma(x) = x^3$. Una condición para la función de activación de la salida podría ser que su imagen fuese todo $\mathbb{R}$ ya que si no, sería imposible aproximar una función que toma valores fuera de la imagen de nuestra salida.

**Ejemplo 2.3.** *Vamos a aproximar el polinomio $p(x) = x^3 + x^2 - x - 1$ en el intervalo $[-2, 2]$ con una arquitectura $(1, 6, 1)$; función de activación $\sigma(x) = \text{máx}\{0, x\}$ (ReLU) y salida la identidad:*

$$W_1 = \left(\begin{array}{c|c} -4.85 & -7.7 \\ -1.25 & -1.3 \\ 1.2 & 1 \\ 1.2 & -0.2 \\ 2 & -1.1 \\ 5 & -5 \end{array}\right) \qquad W_2 = \left(\begin{array}{cccccc|c} -1 & -1 & -1 & 1 & 1 & 1 & 0 \end{array}\right)$$




*La salida final de nuestra red es la función:*

$$F(W,x) = -\sigma(-4.85x-7.77)-\sigma(-1.25x-1.3)-\sigma(1.2x+1)+\sigma(1.2x-0.2)+\sigma(2x-1.1)+\sigma(5x-5)$$

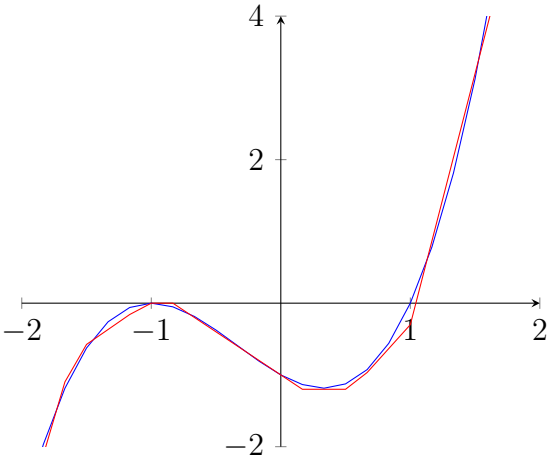



***Figura 2.3:*** *Aproximación de la función* $p(x) = x^3 + x^2 - x - 1$ *mediante una red neuronal con arquitectura* $(1, 6, 1)$ *y función de activación* $R(x) = máx\{0, x\}$.

Aunque hemos dado un resultado muy restrictivo, como por ejemplo la condición de que en la salida tengamos la función identidad, nos sirve para ilustrar que, en efecto, nuestro modelo funciona y sirve para aproximarnos a funciones. Lo que no sabemos todavía es cómo encontrar una arquitectura y pesos adecuados. No existe ninguna fórmula para calcularlos, pero si que existen métodos iterativos, como veremos en el siguiente capítulo, para reajustarlos.

Este capítulo se ha elaborado a partir de los siguientes artículos: [5, 6]. En ellos se responden a más cuestiones. Muchos de los resultados se prueban para las funciones continuas en vez de funciones en $L^2(I^n)$. También se puede encontrar una prueba de que la función $ReLU$ es discriminatoria.

# Capítulo 3

# Algoritmo de propagación hacia atrás

Ya hemos dado una serie de resultados que nos indican que, en efecto, bajo ciertas condiciones, las redes neuronales nos sirven para aproximar funciones, pero todavía nos falta por responder la siguiente pregunta: ¿Cómo la encontramos?

La elección de una arquitectura adecuada no es sencilla, no obstante, lo que si que podemos hacer es encontrar un método de reajuste de pesos (conocido como el *algoritmo de propagación hacia atrás*) con el fin de acercarnos más a nuestra función. A grandes rasgos, lo que hace este algoritmo es tomar un conjunto de entrenamiento y minimizar una función que mide el error cometido por nuestra red en ese conjunto, mediante un proceso iterativo conocido como el *descenso de gradiente*.

## 3.1. Nociones básicas de cálculo diferencial

**Definición 3.1** (Derivada parcial)**.** *Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función diferenciable en $x = (x_1, ..., x_n) \in \mathbb{R}^n$, $n \in \mathbb{N}$. Se define la derivada parcial de $f$ respecto a la componente $x_j$, $j \in \{1, ..., n\}$ como:*

$$\frac{\partial f}{\partial x_j}(x) := \lim_{h \to 0} \frac{f(x + he_j) - f(x)}{h} \tag{3.1}$$

aquí $e_j$ denota el vector canónico en $\mathbb{R}^n$. Notar que este límite esta bien definido ya que $f$ es diferenciable en $x$.

**Definición 3.2** (Gradiente)**.** *Si $f : \mathbb{R}^n \to \mathbb{R}$ es una función diferenciable en $x = (x_1, ..., x_n) \in \mathbb{R}^n$, $n \in \mathbb{N}$, entonces se define el gradiente de $f$ en ese punto como:*

$$\nabla f(x) := \left( \frac{\partial f}{\partial x_1}(x), ..., \frac{\partial f}{\partial x_n}(x) \right) \in \mathbb{R}^n \tag{3.2}$$

**Definición 3.3** (Regla de la cadena)**.** *Sean* $f : \mathbb{R}^n \to \mathbb{R}$ *y* $g : \mathbb{R} \to \mathbb{R}$ *dos funciones diferenciables. Entonces,* $\forall j \in \{1, .., n\}$*:*

$$\frac{\partial}{\partial x_j} f(x_1, ..., g(x_j), ..., x_n) = g'(x_j) \cdot \frac{\partial f}{\partial x_j}(x_1, ..., g(x_j), ..., x_n) \tag{3.3}$$

**Definición 3.4** (Mínimo local)**.** *Sea* $f : \mathbb{R}^n \to \mathbb{R}$*,* $n \in \mathbb{N}$*. El punto* $x^* \in \mathbb{R}^n$ *es un mínimo local si existe* $\epsilon > 0$ *de manera que:*

$$\forall y \in \{x \in \mathbb{R}^n; \|x - x^*\|_2 < \epsilon\}; \;\; f(x^*) \leq f(y)$$

De forma análoga se define el máximo local, además, si esta propiedad se verifica $\forall y \in \mathbb{R}^n$, diremos que $x^*$ es un *mínimo global*.

**Definición 3.5** (Sucesión de convergente)**.** *Sea* $\{x_m\}_{m\in\mathbb{N}} \subset \mathbb{R}^n$ *una sucesión. Diremos que la sucesión converge a* $x^* \in \mathbb{R}^n$ *si para todo* $\epsilon > 0$ *existe un* $m_0 \in \mathbb{N}$ *tal que para todo* $m \geq m_0$*:*

$$\|x_m - x^*\|_2 < \epsilon$$

**Definición 3.6** (Sucesión de Cauchy)**.** *Sea* $\{x_m\}_{m\in\mathbb{N}} \subset \mathbb{R}^n$ *una sucesión. Diremos que es de Cauchy si parar todo* $\epsilon > 0$ *existe un* $m_0 \in \mathbb{N}$ *tal que para todo* $i, j \geq m_0$*:*

$$\|x_i - x_j\|_2 < \epsilon$$

No lo vamos a probar aunque es muy conocido que una sucesión es de Cauchy si y solo si es convergente (por ser $\mathbb{R}^n$ un espacio métrico completo).

## 3.2. Descenso por gradiente

Dada una función $f : \mathbb{R}^n \to [0, \infty)$, $n \in \mathbb{N}$, diferenciable en todo su dominio, se define la sucesión de descenso por gradiente para $f$ como:

$$\begin{cases} x_1 \in \mathbb{R}^n \\ x_{m+1} = x_m - \alpha \nabla f(x_m) \end{cases} \tag{3.4}$$

con $\alpha = \alpha(f) > 0$ el *ratio de aprendizaje* a determinar. Hemos tomado como dominio de nuestra función $f$ el intervalo $[0, \infty)$, ya que este algoritmo lo vamos a utilizar para funciones que miden cierto error cometido y por tanto, son positivas.

La idea consiste en determinar un $\alpha$ de manera que la sucesión de descenso por gradiente $\{x_m\}_{m\in\mathbb{N}}^{\alpha}$ converja hacia un cierto mínimo local $x^*$. Una variación de este método podría ser modificando el ratio de aprendizaje $\alpha$ en cada iteración, esto es, si $\{\alpha_m\}_{m\in\mathbb{N}} \subset (0,\infty)$:

$$\begin{cases} x_1 \in \mathbb{R}^n \\ x_{m+1} = x_m - \alpha_m \nabla f(x_m) \end{cases} \tag{3.5}$$

**Lema 3.1.** *Sea $\{x_m\}_{m\in\mathbb{N}}^{\alpha_m} \subset \mathbb{R}^n$ una sucesión de descenso por gradiente de una función diferenciable $f : \mathbb{R}^n \to [0,\infty)$, $n \in \mathbb{N}$. Entonces, para cada $m \in \mathbb{N}$, existe $\alpha_m > 0$ verificando:*

$$f(x_{m+1}) = f(x_m - \alpha_m \nabla f(x_m)) \leq f(x_m)$$

*Demostración.* Consideramos la función auxiliar:

$$\psi(\alpha) = f(x_{m+1}) = f(x_m - \alpha \nabla f(x_m)); \;\; \alpha \in \mathbb{R}$$

notar que aquí estamos permitiendo que $\alpha$ sea negativo. Como $f$ es diferenciable, $\psi$ también, además, calculando su polinomio de Taylor centrado en $\alpha = 0$ tenemos:

$$\psi(\alpha) = \psi(0) + \psi'(0) \cdot \alpha + o(\alpha)$$

$\psi(0) = f(x_m)$. Para calcular $\psi'(0)$ debemos aplicar la regla de la cadena, si $x_m = (x_{m,1}, ..., x_{m,n})$:

$$\psi(\alpha) = f\left(x_{m,1} - \alpha \frac{\partial f}{\partial x_1}(x_m), ..., x_{m,n} - \alpha \frac{\partial f}{\partial x_n}(x_m)\right)$$

por tanto:

$$\psi'(\alpha) = -\sum_{j=1}^{n} \frac{\partial f}{\partial x_j}(x_m) \cdot \frac{\partial f}{\partial x_j}(x_m - \alpha \nabla f(x_m))$$

de donde se deduce:

$$\psi'(0) = -\sum_{j=1}^{n} \frac{\partial f}{\partial x_j}(x_m) \cdot \frac{\partial f}{\partial x_j}(x_m) = -\sum_{j=1}^{n} (\frac{\partial f}{\partial x_j}(x_m))^2 = -\|\nabla f(x_m)\|_2^2$$

juntando estas dos cosas llegamos a que:

$$\psi(\alpha) = f(x_{m+1}) = f(x_m - \alpha \nabla f(x_m)) = f(x_m) - \alpha \cdot \|\nabla f(x_m)\|_2^2 + o(\alpha) > 0$$

tomamos $\alpha = \alpha_m$ suficientemente pequeño y tendremos probado el resultado.

□

Bajo ciertas condiciones se puede encontrar un $\alpha$ global que cumpla la condición del Lema 3.1. para todo $m \in \mathbb{N}$.

**Proposición 3.2.** *Sea $f : X \subset \mathbb{R}^n \to [0, \infty)$ una función diferenciable y $\{x_m\}_{m\in\mathbb{N}}^{\alpha}$ una sucesión de descenso por gradiente para $f$ verificando:*

1. $f(x_{m+1}) \leq f(x_m)$; $\forall m \in \mathbb{N}$.
2. *Existen $C, c > 0$ tal que $\|x_i - x_j\|_2 < C \cdot |f(x_i) - f(x_j)|^c$.*

*entonces, se tiene que la sucesión es convergente y además:*

$$\lim_{m\to\infty} \nabla f(x_m) = \bar{0} \in \mathbb{R}^n$$

*Demostración.* Tenemos que la sucesión $\{f(x_m)\}_{m\in\mathbb{N}} \subset [0, f(x_1)]$ es monótona y acotada, luego es convergente y por lo tanto es de Cauchy.

Tomamos $\epsilon > 0$ y aplicamos a $\{f(x_m)\}_{m\in\mathbb{N}}$ la definición de sucesión de Cauchy para $\sqrt[c]{\epsilon}/C$. Existirá un $m_0 \in \mathbb{N}$ tal que para todo $i, j \geq m_0$:

$$\|x_i - x_j\|_2 < C \cdot |f(x_i) - f(x_j)|^c < C \cdot \frac{\epsilon}{C} = \epsilon$$

por lo tanto, $\{x_m\}_{m\in\mathbb{N}}$ es de Cauchy, luego convergente. Denotamos $x^*$ como su límite y se tendrá:

$$\lim_{m\to\infty} \nabla f(x_m) = \lim_{m\to\infty} \frac{1}{\alpha} \cdot (x_m - x_{m+1}) = \frac{1}{\alpha} \cdot (x^* - x^*) = \bar{0} \in \mathbb{R}^n$$

□

Este resultado no nos garantiza la convergencia a un mínimo local de la función ya que existen otros puntos donde el gradiente se anula y no son ni máximos ni mínimos como los *puntos de silla*.

**Ejemplo 3.1.** *Vamos a estimar los mínimos de la función $f(x) = x(x-1)(x-\frac{3}{2})(x-2)+\frac{1}{2}$.*

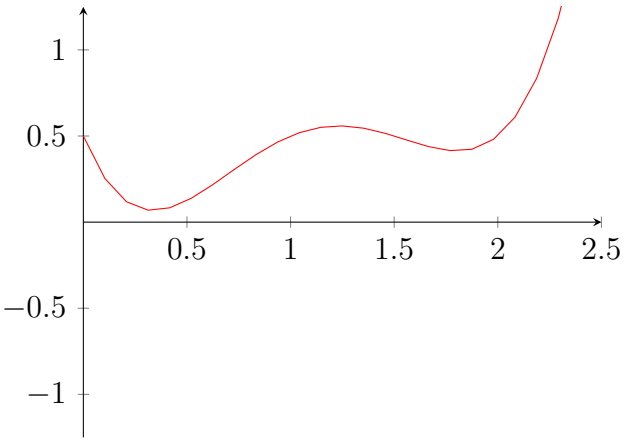



***Figura 3.1:*** *Gráfica de la función $f$ restringida a $(0, \frac{5}{2})$.*

*La función tiene dos mínimos. Tomamos la sucesión habitual con* $\alpha = \frac{1}{10}$*:*

$$\begin{cases} x_1 \in \mathbb{R} \\ x_{n+1} = x_n - \frac{1}{10} \cdot f'(x_n) = -\frac{2}{5} \cdot x_n^3 + \frac{27}{20} \cdot x_n^2 - \frac{3}{10} \cdot x_n + \frac{3}{10} \end{cases}$$

*Para* $x_1 = 0$ *o* $x_1 = 2$*, se puede demostrar que se cumplen las hipótesis de la proposición anterior, además, si* $x_1 = 0$ *la sucesión converge hacia el mínimo global de* $f$ *y que si* $x_1 = 2$*, entonces converge hacia el otro mínimo de* $f$*, esta vez local. De la proposición anterior se deduce que:*

$$\lim_{n\to\infty} f'(x_n) = 0$$

*Mínimo global:* $x_1^* \approx 0.33683$*,* $f(x_1^*) \approx 0.006787$*.*
*Mínimo local:* $x_2^* \approx 1.80363$*,* $f(x_2^*) \approx 0.41358$*.*

En este ejemplo es muy fácil identificar cuáles son los mínimos de la función y cuál de todos ellos es un mínimo global. Cuando el número de variables aumenta esto ya no se puede visualizar, por lo que la elección de un $x_1$ adecuado se hace más difícil ya que no sabemos si la sucesión va a converger a un mínimo local o uno global (o incluso algún otro caso). Recordar que vamos a intentar minimizar un cierto error cometido y por tanto va a ser interesante evitar caer en mínimos locales.

## 3.3. Propagación hacia atrás

Sea $g^* : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ una aplicación, consideramos una arquitectura arbitraria $(n_0, .., n_l)$ con sus pesos y umbrales $W \in \mathcal{W}$, su función de activación derivable $\sigma$ y su función de salida:

$$F(W, \cdot) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$$

Tomamos un conjunto de entrenamiento de $T$ elementos: $\{x_t, g^*(x_t)\}_{t\leq T}$ y medimos el error cometido por nuestra red mediante la siguiente función:

$$\begin{aligned} \mathcal{J} &: \mathcal{W} \to [0, \infty) \\ W &\longmapsto \sum_{t=1}^{T} \mathcal{E}\left(F\left(W, x_t\right), g^*(x_t)\right) \end{aligned} \tag{3.6}$$

donde $\mathcal{E} : \mathbb{R}^{n_l} \times \mathbb{R}^{n_l} \to [0, \infty)$ es una función que mide el error cometido en cada entrada. Además, deberá ser diferenciable respecto a su primer argumento y verificar que:

$$\mathcal{E}(x, x) = 0; \ \ \forall x \in \{g^*(x_t)\}_{t\leq T}$$

El objetivo será minimizar nuestra función $\mathcal{J}$ mediante el método del descenso por gradiente definido en la sección anterior, con pequeñas modificaciones ya que las funciones que hemos considerado tenían su dominio en $\mathbb{R}^n$, en cambio, ahora queremos minimizar una función cuyo dominio está en $\mathcal{W}$.

**Ejemplo 3.2.** *(Algunas funciones de error) Nuestra función de error $\mathcal{J}$ queda determinada cuando hayamos elegido cual es nuestra función $\mathcal{E}$.*

*La elección de $\mathcal{E}$ es muy importante y depende de las características de nuestro problema. Vamos a poner algunos ejemplos:*

$$\mathcal{E}(x, y) = \|x - y\|_2^2$$

$$\mathcal{E}(x, y) = \sum_{i=1}^{n_l} (log(1 + x_i) - log(1 + y_i))^2$$

*estas funciones, independientemente de que sean o no las mejores sirven para cualquier tipo de problema ya que verifican:*

$$\mathcal{E}(x, x) = 0; \ \ \forall x \in \mathbb{R}^{n_l}$$

*Otro ejemplo, si tenemos un problema de clasificación de $n_l$ clases diferentes se suele tomar la función:*

$$\mathcal{E}(x, y) = -\sum_{j=1}^{n_l} log(x_j) \cdot y_j$$

*aquí suponemos $x$ como la salida proporcionada por la red e $y$ el valor real. Para evitar los problemas que nos da el logaritmo y conseguir que $\mathcal{E} > 0$ podemos hacer que $x \in (0, 1)^{n_l}$ tomando como función de activación en $c_l$, por ejemplo, la función logística, o la softmax (que es la que se usa para este tipo de problemas). Por ejemplo, si $y = e_4$ (la clase 4) Tenemos:*

$$\mathcal{E}(x, e_4) = -log(x_4)$$

*como $x_4 \in (0, 1)$ cuanto más próximo a 1 nos encontremos, más pequeña será nuestra función $\mathcal{E}$. Además, en el caso de que tengamos la función softmax en la salida, los demás valores estarán próximos a cero y en consecuencia $x \approx e_4$ ya que:*

$$x_1 + ... + x_{n_l} = 1$$

*Otra función parecida que nos puede servir como alternativa puede ser:*

$$\mathcal{E}(x, y) = \sum_{j=1}^{n_l} (1 - x_i) \cdot y_j$$

Ahora vamos a mostrar un ejemplo para ilustrar cómo podemos usar los resultados que hemos visto en la sección anterior, para funciones que esta vez no tienen como dominio $\mathbb{R}^n$, sino nuestro conjunto $\mathcal{W}$:

**Ejemplo 3.3.** *(Cambio en la notación) Consideramos las siguientes funciones:*

$$(x_1, x_2, x_3, x_4) \to f(x_1, x_2, x_3, x_4)$$

$$\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix} \to f(x_1, x_2, x_3, x_4)$$

*Notar que en el primer caso, estamos tomando como dominio de nuestra función $\mathbb{R}^4$, en el segundo caso, estamos tomando como dominio $M_{2\times 2}(\mathbb{R})$. Sabemos que estos dos espacios vectoriales son isomorfos, por lo tanto, no importa cuál de ellos usemos ya que podemos interpretarlo simplemente como un cambio en la notación. El gradiente de $f$ en el primer caso es el vector:*

$$\left(\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \frac{\partial f}{\partial x_3}(x), \frac{\partial f}{\partial x_4}(x)\right)$$

*por lo que tiene sentido definir el gradiente para $f$ en el segundo caso como:*

$$\begin{pmatrix} \frac{\partial f}{\partial x_1}(x) & \frac{\partial f}{\partial x_2}(x) \\ \frac{\partial f}{\partial x_3}(x) & \frac{\partial f}{\partial x_4}(x) \end{pmatrix}$$

La idea que hemos aplicado en este ejemplo, la vamos a llevar a lo grande, tomando como espacio de salida nuestro $\mathcal{W}$, para ello recordar que:

$$\mathcal{W} = M_{n_1\times n_0+1}(\mathbb{R}) \times ... \times M_{n_l\times n_{l-1}+1}(\mathbb{R})$$

así, un elemento de $\mathcal{W}$ es un vector de matrices. Podemos sumar dos elementos de $\mathcal{W}$ componente a componente usando en cada una de ellas la suma habitual de las matrices:

$$(W_1, ..., W_l) + (W_1', ..., W_l') = (W_1 + W_1', ..., W_l + W_l')$$

por tanto, podemos considerar $\mathcal{W}$ como un espacio vectorial sobre $\mathbb{R}$ de dimensión:

$$dim(\mathcal{W}) := N = \sum_{i=1}^{l} (n_{i-1} + 1) \cdot n_i$$

Este espacio es isomorfo a $\mathbb{R}^N$, además, podemos pensar que simplemente estamos haciendo un cambio de notación, de forma adecuada, para que nuestros pesos estén agrupados en orden según su capa y su neurona.

**Definición 3.7** (Gradiente en $\mathcal{W}$)**.** *Sea $f : \mathcal{W} \to [0, \infty)$ una aplicación diferenciable para cada peso y umbral. Dado $W = (W_1, ..., W_l) \in \mathcal{W}$, definimos el gradiente de $f$ en el vector $W$ como:*

$$\nabla f(W) := \left( \frac{\partial f}{\partial W_1}(W), ..., \frac{\partial f}{\partial W_l}(W) \right) \in \mathcal{W} \tag{3.7}$$

*donde para cada $j \in \{1, ..., l\}$, tenemos la matriz:*

$$\frac{\partial f}{\partial W_j}(W) = \left( \frac{\partial f}{\partial w_{i,k}^j}(W) \,\middle|\, \frac{\partial f}{\partial b_i^j}(W) \right)_{i \leq n_j, k \leq n_{j-1}}$$

**Ejemplo 3.4.** *Tomamos $\mathcal{W} = M_{2\times 3}(\mathbb{R}) \times M_{2\times 3}(\mathbb{R})$ y la función que a cada $W = (W_1, W_2) \in \mathcal{W}$ le asigna:*

$$f(W) = \frac{1}{2} \sum_{j=1}^{2} \sum_{i,k=1}^{2} \left[ (w_{i,k}^j)^2 + (b_i^j)^2 \right]$$

*entonces:*

$$\nabla f(W) = \left( \left( \begin{array}{cc|c} w_{1,1}^1 & w_{1,2}^1 & b_1^1 \\ w_{2,1}^1 & w_{2,2}^1 & b_2^1 \end{array} \right), \left( \begin{array}{cc|c} w_{1,1}^2 & w_{1,2}^2 & b_1^2 \\ w_{2,1}^2 & w_{2,2}^2 & b_2^2 \end{array} \right) \right) = (W_1, W_2) = W$$

*Notar que aquí los superíndices solo indican a qué componente del vector pertenecen los pesos, no son potencias.*

Vemos que la notación es muy engorrosa, pero hay que entender que estamos hablando de muchas variables; además, es una buena forma de organizar nuestros pesos y umbrales por capas y neuronas. Tenemos que minimizar nuestra función $\mathcal{J}$, para ello, vamos a aplicarle un método de reajuste de pesos, conocido como el algoritmo de propagación hacia atrás, el cual consiste simplemente en aplicar un descenso por gradiente de forma análoga a la sección anterior (esta vez con el gradiente en $\mathcal{W}$):

$$\begin{cases} W^{(1)} \in \mathcal{W} \\ W^{(n+1)} = W^{(n)} - \alpha \nabla \mathcal{J}(W^{(n)}) \end{cases} \tag{3.8}$$

Los resultados de convergencia son idénticos ya que, como hemos dicho, podemos tratar nuestro $\mathcal{W}$ como un $\mathbb{R}^N$ ordenado de cierta forma. La actualización de los pesos será más compleja conforme aumentemos el número de capas de la red y/o el número de neuronas en cada capa. Algunas de las dificultades de este método son:

1. No siempre es fácil estimar un ratio de aprendizaje $\alpha$ para una correcta convergencia.

2. El cálculo de las derivadas no es sencillo y es preciso aplicar la regla de la cadena varias veces. De hecho, cuanto más alejados estén nuestros pesos de la capa de salida, más veces habrá que aplicar la regla de la cadena (esto es consecuencia de que la función salida $F(W, \cdot)$ se obtiene al componer todas las funciones de transición entre capas).

3. Aunque nuestro algoritmo converja, no sabemos si dicho límite es un *mínimo global*. Quizá este sea el mayor impedimento de todos, ya que no podemos saber a ciencia cierta que $W^{(1)}$ inicial tomar para alcanzarlo.

Vamos a ver cómo se actualizan los pesos de la capa $c_j$, $j \in \{1, ..., l\}$:

$$W_j^{(n+1)} = W_j^{(n)} - \alpha \frac{\partial \mathcal{J}}{\partial W_j}(W^{(n)}) \Rightarrow \begin{cases} (w_{i,k}^j)^{(n+1)} = (w_{i,k}^j)^{(n)} - \alpha \frac{\partial \mathcal{J}}{\partial w_{i,k}^j}(W^{(n)}) \\ \\ (b_i^j)^{(n+1)} = (b_i^j)^{(n)} - \alpha \frac{\partial \mathcal{J}}{\partial b_i^j}(W^{(n)}) \end{cases} \tag{3.9}$$

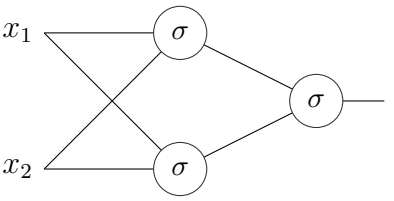
esto es, el peso $k-$ésimo de la neurona $i-$ésima en la capa $c_j$ se actualiza siguiendo esta regla 3.9, de forma análoga con el umbral, $k \in \{1, ..., n_{j-1}\}$ e $i \in \{1, ..., n_j\}$. La complejidad de este método reside en el cálculo de las expresiones:

$$\frac{\partial \mathcal{J}}{\partial w_{i,k}^j} \qquad \frac{\partial \mathcal{J}}{\partial b_i^j}$$

ya que, como hemos dicho, deberá ser preciso aplicar la regla de la cadena varias veces, cada vez más según nos vayamos alejando de la capa de salida $c_l$.

**Ejemplo 3.5.** *Vamos a aproximarnos a la función lógica XOR aplicando este método. Tomamos como conjunto de entrenamiento sus* 4 *salidas, recuperamos la arquitectura* $(2, 2, 1)$ *propuesta en el ejemplo 1.5 y tomamos como función de activación la función logística en todas nuestras capas (incluida la salida).*

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



$$W = \left( \left( \begin{array}{cc|c} w^1_{1,1} & w^1_{1,2} & b^1_1 \\ w^1_{2,1} & w^1_{2,2} & b^1_2 \end{array} \right), \left( \begin{array}{cc|c} w^2_{1,1} & w^2_{1,2} & b^2_1 \end{array} \right) \right) = (W_1, W_2)$$

$$F(W,(x,y)) = \sigma(w^2_{1,1} \cdot \sigma(w^1_{1,1} \cdot x_1 + w^1_{1,2} \cdot x_2 - b^1_1) + w^2_{1,2} \cdot \sigma(w^1_{2,1} \cdot x_1 + w^1_{2,2} \cdot x_2 - b^1_2) - b^2_1)$$

$$\mathcal{J}(W) = (F(W,(0,0)) - 0)^2 + (F(W,(1,1)) - 0)^2 + (F(W,(0,1)) - 1)^2 + (F(W,(1,0)) - 1)^2$$

*Notar que hemos tomado* $\mathcal{E}(x,y) = (x-y)^2$*. En este caso es imposible lograr una aproximación exacta, pero podemos repetir el algoritmo hasta, por ejemplo, obtener un error* $\mathcal{J}(W) < 0.01$*. Vamos a tomar un ratio de aprendizaje* $\alpha = 1$ *y pesos iniciales:*

$$W^{(1)} = \left( \left( \begin{array}{cc|c} -3 & 1 & 1 \\ 1 & -3 & 1 \end{array} \right), \left( \begin{array}{cc|c} 1 & 2 & 1 \end{array} \right) \right) = (W_1, W_2)$$

*tras* 115 *iteraciones del algoritmo 3.8 obtenemos que* $\mathcal{J}(W^{(115)}) \approx 0.0498$ *y después de iterar* 500 *veces* $\mathcal{J}(W^{(500)}) \approx 0.0067$*.*

$$W^{(500)} \approx \left( \left( \begin{array}{cc|c} -5.464 & 5.239 & 2.940 \\ 5.291 & -5.470 & 2.986 \end{array} \right), \left( \begin{array}{cc|c} 7.700 & 7.700 & 3.800 \end{array} \right) \right)$$

*en cambio, si tomamos* $W^{(1)}$ *como todo unos, tras* 3000 *iteraciones, se tiene que* $\mathcal{J}(W^{(3000)}) \approx 0.6688$*, quedándose estancado en un mínimo local o un punto de silla.*

Este ejemplo es interesante. En la Proposición 3.2 dábamos unas condiciones que nos garantizaban que tanto $\{W^{(n)}\}_{n\in\mathbb{N}}$ como $\{\mathcal{J}(W^{(n)})\}_{n\in\mathbb{N}}$ convergían. Para el primer caso, vemos que podemos hacer el error tan pequeño como queramos, pero la convergencia no se da a un mínimo global ya que eso implicaría que existen $\xi_1, \xi_2 \in \mathbb{R}$ tal que $\sigma(\xi_1) = 0$ y $\sigma(\xi_2) = 1$ lo cual es absurdo porque esos valores solo se alcanzan en el límite. En este caso, solamente se da la convergencia a cero de $\{\mathcal{J}(W^{(n)})\}_{n\in\mathbb{N}}$ (El gradiente si que va acercándose a cero pero hacia una zona asintótica donde algunos pesos se van a $\pm\infty$)[1].

Vemos que la correcta aproximación ha dependido del ratio de aprendizaje que hemos tomado y sobre todo de los pesos iniciales. En uno de los casos hemos conseguido aproximarnos correctamente mientras que en el otro no. Como no podemos confiar en una correcta elección de nuestro punto de partida, en la siguiente sección explicaremos una variación del algoritmo de propagación hacia atrás ya explicado. Así, a la hora de construir un modelo de

[1]Por ejemplo, no podemos decir que $\pm\infty$ sea un mínimo global de la función $1/x^2$, pero podemos tomar un $M \in \mathbb{R}$ tan grande como nosotros queramos para que $1/M^2$ sea pequeño. También se tiene que su derivada $-2/M^3$ es tan pequeña como nosotros queramos.

red neuronal, podremos disponer de varias alternativas para minimizar la función de error $\mathcal{J}$.

Por último comentar que para la realización del Ejemplo 3.5. se ha utilizado la plataforma de SAGE que nos proporciona la Universidad de La Rioja.

## 3.4. Optimizadores

Dada $f$ diferenciable y $\alpha > 0$, recordar que la sucesión de descenso por gradiente se definía del siguiente modo:

$$\begin{cases} x_1 \in \mathbb{R}^n \\ x_{m+1} = x_m - \alpha \nabla f(x_m) \end{cases}$$

el parámetro $\alpha$ era nuestro ratio de aprendizaje. Lo que se propone ahora es usar otro parámetro $\beta \geq 0$ a el cual lo llamaremos *momento* y definimos la siguiente sucesión $\{(x_m, z_m)\}_{m\in\mathbb{N}}^{\alpha,\beta}$:

$$\begin{cases} x_1, z_1 \in \mathbb{R}^n \\ z_{m+1} = \beta z_m + \nabla f(x_m) \\ x_{m+1} = x_m - \alpha z_{m+1} \end{cases} \tag{3.10}$$

para $\beta = 0$ recuperamos nuestra sucesión de descenso por gradiente $\{x_m\}_{m\in\mathbb{N}}^{\alpha}$. La idea viene a ser la misma otra vez, hay que determinar $\alpha$ y $\beta$ adecuados para conseguir minimizar nuestra función $f$. Una condición necesaria es $\beta < 1$, esto se debe a que queremos que la sucesión $\{x_m\}_{m\in\mathbb{N}}$ converja, lo que implica (si $\alpha > 0$):

$$\lim_{m\to\infty} z_{m+1} = 0$$

por otro lado:

$$z_{m+1} = \beta z_m + \nabla f(x_m) = ... = \beta^m \cdot \left( z_1 + \sum_{k=1}^{m} \beta^{-k} \nabla f(x_k) \right)$$

de donde se deduce que si $\beta \geq 1$, $z_{m+1}$ no tiende a cero (como normal general). Por tanto, a la hora de estimar nuestros parámetros debemos saber que $\alpha \in (0, \infty)$ y $\beta \in [0, 1)$.

En términos de nuestro conjunto $\mathcal{W}$ y nuestra función de error $\mathcal{J}$ denotamos este algoritmo como:

$$\begin{cases} W^{(1)}, W'^{(1)} \in \mathcal{W} \\ W'^{(n+1)} = \beta W'^{(n)} + \nabla \mathcal{J}(W^{(n)}) \\ W^{(n+1)} = W^{(n)} - \alpha W'^{(n+1)} \end{cases} \tag{3.11}$$

Tanto el algoritmo de propagación hacia atrás como este que acabamos de proponer ahora añadiendo un momento $\beta$ son solo dos casos particulares de lo que se conoce como un **optimizador** de nuestra función error $\mathcal{J}$. Demos una definición más formal:

**Definición 3.8.** *(Optimizador) Sea $f : \mathbb{R}^n \to [0, \infty)$ una función diferenciable. Un optimizador de $f$ es una sucesión $\{x_m\}_{m\in\mathbb{N}}$ tal que:*

$$\exists \lim_{m\to\infty} x_m := x^* \Rightarrow \nabla f(x^*) = \bar{0} \in \mathbb{R}^n$$

En otras palabras, si la sucesión es convergente, lo hace a un punto donde el gradiente es cero. Así, tenemos varias alternativas a elegir con el fin de evitar estancamientos a la ahora de entrenar nuestra red neuronal artificial.

Al igual que pasa con nuestra función de error $\mathcal{J}$, existen muchos tipos de optimizadores, más o menos útiles dependiendo de las características de nuestro problema. De hecho, es fácil construir este tipo de sucesiones, por ejemplo:

$$\begin{cases} x_1 \in \mathbb{R}^n \setminus \{0\} \\ x_{m+1} = x_m - \|\nabla f(x_m)\|_2 \cdot x_1 \end{cases}$$

es un optimizador de la función $f$ ya que si la sucesión es convergente a $x^*$ entonces:

$$x^* = x^* - \|\nabla f(x^*)\|_2 \cdot x_1 \Rightarrow \|\nabla f(x^*)\|_2 = 0 \in \mathbb{R} \Rightarrow \nabla f(x^*) = \bar{0} \in \mathbb{R}^n$$

Cabe recalcar que no hay ningún optimizador que sea mejor que otro. Dependerá de cada problema y de sus características. El optimizador más conocido es el de descenso por gradiente y de ahí que sea el que hayamos explicado con profundidad. No obstante, hay muchos artículos donde se tratan cosas muy interesantes como por ejemplo, la búsqueda de cotas para estimar la velocidad de convergencia (por ejemplo, ver bajo que condiciones cierto optimizador es una $o(1/n^2)$).

En la siguiente plataforma podemos encontrar bastantes optimizadores: [7]. Aquí hay dos artículos donde se explican algunos (el segundo es el más matemático): [8, 9].

Por último, comentar que este capítulo se ha elaborado a partir de: [5, 10, 11].

# Capítulo 4

# Convergencia hacia un mínimo global

Los tres primeros capítulos constituyen la primera parte de nuestro trabajo, en ellos hemos dado una serie de resultados que nos indicaban que nuestro modelo funciona y es capaz de enfrentarse a los problemas de aprendizaje automático.

En este capítulo y como segunda parte de nuestro trabajo, vamos a tratar uno de los problemas que hay actualmente sobre este modelo de red neuronal, que es el de encontrar ciertas garantías que nos permitan determinar cuándo un mínimo es global, ya que como vimos, el algoritmo de propagación hacia atrás convergía hacia un punto donde el gradiente era cero.

## 4.1. Condición de convexidad

Vamos a suponer que nuestra función de error $\mathcal{J}$ satisface la condición de convexidad.

**Definición 4.1** (función convexa)**.** *Una función $f : \mathbb{R}^n \to \mathbb{R}$ se dice convexa si para todo $x, y \in \mathbb{R}^n$ se tiene:*

$$f(\lambda \cdot x + (1-\lambda) \cdot y) \leq \lambda \cdot f(x) + (1-\lambda) \cdot f(y); \ \ \forall \lambda \in [0,1]$$

Cuando una función es lineal notar que entonces se da la igualdad. En nuestro conjunto de pesos $\mathcal{W}$, la función error $\mathcal{J}$ será convexa si para todo $W, W' \in \mathcal{W}$ se verifica:

$$\mathcal{J}(\lambda \cdot W + (1-\lambda) \cdot W') \leq \lambda \cdot \mathcal{J}(W) + (1-\lambda) \cdot \mathcal{J}(W'); \ \ \forall \lambda \in [0,1]$$

**Teorema 4.1.** *Si $f : \mathbb{R}^n \to \mathbb{R}$ es una función convexa y $x^* \in \mathbb{R}^n$ un mínimo local de $f$ entonces necesariamente debe ser un mínimo global.*

*Demostración.* Vamos a buscar una contradicción suponiendo que $x^*$ no es un mínimo global.

Sea $y^* \in \mathbb{R}^n$ verificando que $f(y^*) < f(x^*)$. Por ser $f$ convexa, dado $\lambda \in (0,1)$:

$$f(\lambda \cdot x^* + (1-\lambda) \cdot y^*) \leq \lambda \cdot f(x^*) + (1-\lambda) \cdot f(y^*) < \lambda \cdot f(x^*) + (1-\lambda) \cdot f(x^*) = f(x^*)$$

Como $x^*$ es un mínimo local, existirá $\epsilon > 0$ de manera que:

$$\forall y \in \{x \in \mathbb{R}^n; \|x - x^*\|_2 < \epsilon\} : \ f(x^*) \leq f(y)$$

este $\epsilon$ no es único ya que $\epsilon' < \epsilon$ también cumple esa condición. Por tanto, tomamos $\epsilon'$ de la siguiente manera:

$$0 < \epsilon' < mín\{\|y^* - x^*\|_2, \epsilon\}$$

que es mayor que cero ya que $\epsilon > 0$ y $f(y^*) \neq f(x^*) \Rightarrow \|y^* - x^*\|_2 > 0$. Ahora, para ese $\epsilon'$ vamos a tomar $\lambda$ tal que:

$$0 < 1 - \lambda < \frac{\epsilon'}{\|y^* - x^*\|_2}$$

notar que $0 < 1 - \lambda < 1 \Rightarrow 0 < \lambda < 1$. Para ese $\lambda$:

$$\|\lambda \cdot x^* + (1-\lambda) \cdot y^* - x^*\|_2 = \|(\lambda - 1) \cdot x^* + (1-\lambda) \cdot y^*\|_2 = (1-\lambda) \cdot \|y^* - x^*\|_2 < \epsilon' < \epsilon$$

y por tanto:

$$\lambda \cdot x^* + (1-\lambda) \cdot y^* \in \{x \in \mathbb{R}^n; \|x - x^*\|_2 < \epsilon\} \ \Rightarrow \ f(\lambda \cdot x^* + (1-\lambda) \cdot y^*) \geq f(x^*)$$

obtenemos una contradicción ya que previamente habíamos probado por la convexidad de $f$ que se cumplía justamente la desigualdad contraria.

□

## 4.2. Condición de rango máximo

Bajo ciertas condiciones, podemos encontrar más formas de saber cuándo un punto crítico ($\nabla \mathcal{J}(W^*) = 0$) es un mínimo global. En concreto, vamos a dar una forma de caracterizar nuestros puntos críticos en una red neuronal con una arquitectura igual a la propuesta en la Figura 1.4. Esta caracterización se puede extender de forma análoga a cualquier red neuronal hacia delante sin la condición de que la función de activación en todas las capas ocultas deba ser la misma. Recordar que en el capítulo anterior habíamos definido para una arquitectura $(n_0, ..., n_l)$:

$$dim(\mathcal{W}) := N = \sum_{i=1}^{l} (n_{i-1} + 1) \cdot n_i$$

además, se tenía que $\mathcal{W} \simeq \mathbb{R}^N$ (son espacios vectoriales isomorfos). Para todo $W \in \mathcal{W}$, $\nabla\mathcal{J}(W) \in \mathcal{W}$. Podemos construir la siguiente aplicación:

$$\begin{aligned} \wedge\colon \mathcal{W} &\to \mathbb{R}^N \\ W &\longmapsto \hat{W} \end{aligned}$$

para transformar nuestros elementos de $\mathcal{W}$ (vectores de matrices) en vectores de $\mathbb{R}^N$. La transformación consiste en poner en fila todas las filas de nuestras matrices, de la primera a la última. Por ejemplo:

$$W = \left( \left( \begin{array}{cc|c} w^1_{1,1} & w^1_{1,2} & b^1_1 \\ w^1_{2,1} & w^1_{2,2} & b^1_2 \end{array} \right), \left( \begin{array}{cc|c} w^2_{1,1} & w^2_{1,2} & b^2_1 \end{array} \right) \right) = (W_1, W_2)$$

entonces:

$$\hat{W} = \left( w^1_{1,1}, w^1_{1,2}, b^1_1, w^1_{2,1}, w^1_{2,2}, b^1_2, w^2_{1,1}, w^2_{1,2}, b^2_1 \right)$$

de hecho, esta aplicación es uno de los posibles isomorfismos entre estos dos espacios vectoriales. Con esta notación, vamos a ser capaces de plantear $\hat{\nabla\mathcal{J}}(\cdot) \in \mathbb{R}^N$ como un sistema de ecuaciones. Por lo tanto, los puntos críticos de nuestra función error, se podrán ver como las soluciones triviales de ese sistema.

$$\nabla\mathcal{J}(W) = 0 \in \mathcal{W} \Leftrightarrow \hat{\nabla\mathcal{J}}(W) = \bar{0} \in \mathbb{R}^N$$

Para plantear $\hat{\nabla\mathcal{J}}(\cdot)$ como un sistema de ecuaciones hay que realizar una serie de operaciones que no vamos a explicar con todo el detalle. El desarrollo de las mismas se puede encontrar en [12] y en [13] de forma más simplificada. Estas operaciones no dejan de ser una manera de estructurar todas las derivadas aplicando la regla de la cadena a nuestra función de error $\mathcal{J}$.

**Definición 4.2.** *(Producto de Kronecker) Sean $A \in M_{m\times n}(\mathbb{R})$ y $B \in M_{p\times q}(\mathbb{R})$ dos matrices. Se define el producto de Kronecker de $A$ por $B$ y se denota como $A \otimes B$ a la siguiente matriz por bloques:*

$$A \otimes B = \begin{pmatrix} a_{1,1} \cdot B & \cdots & a_{1,n} \cdot B \\ \vdots & \ddots & \vdots \\ a_{m,1} \cdot B & \cdots & a_{m,n} \cdot B \end{pmatrix} \in M_{m\cdot p \times n \cdot q}$$

Es fácil ver que esta operación no es conmutativa. Por ejemplo:

$$\begin{pmatrix}1 & 2\end{pmatrix} \otimes \begin{pmatrix}3 & 1\\ 1 & 1\end{pmatrix} = \begin{pmatrix}3 & 1 & 6 & 2\\ 1 & 1 & 2 & 2\end{pmatrix}$$

$$\begin{pmatrix}3 & 1\\ 1 & 1\end{pmatrix} \otimes \begin{pmatrix}1 & 2\end{pmatrix} = \begin{pmatrix}3 & 6 & 1 & 2\\ 1 & 2 & 1 & 2\end{pmatrix}$$

Vamos a tomar una arquitectura cualquiera $(n_0, ..., n_l)$ con función de activación $\sigma$ y una salida $y$, una función a aproximar $g^*$ y un conjunto de entrenamiento de $T$ elementos $\{(x_t, g^*(x_t))\}_{t\leq T} \subset \mathbb{R}^{n_0}$. Para cada $t$, vamos a denotar:

$$\begin{cases} \phi_0(x_t) = (x_t, -1) \in \mathbb{R}^{n_0+1} \\ \phi_j(x_t) = (\sigma_j \circ ... \circ \sigma_1(x_t), -1) \in \mathbb{R}^{n_j+1} \end{cases} \tag{4.1}$$

recordar que las $\sigma_i$ eran las funciones de transición definidas en el Sección 1.4. Simplemente lo que estamos haciendo es devolver la salida de la capa $c_j$ en vez de la salida final de la red (proporcionada por la capa $c_l$). Además, le estamos añadiendo una componente más a cada vector con un valor igual a $-1$. Esto se debe a que estamos intentando organizar nuestras derivadas y ese $-1$ va a hacer referencia a la derivada de nuestra función $F(W, x_t)$ respecto a un umbral, que siempre es $-1$.

Ahora vamos a definir las siguientes funciones:

$$\begin{aligned} \sigma_j'(W_j, \cdot) : \mathbb{R}^{n_{j-1}} &\to \mathbb{R}^{n_j} \\ x &\longmapsto \sigma_j'(W_j, x) \end{aligned}$$

$$\sigma_j'(W_j, x) = \sigma' \left( \begin{pmatrix} w^j_{1,1} & \cdots & w^j_{1,n_{j-1}} \\ \vdots & \ddots & \vdots \\ w^j_{n_j,1} & \cdots & w^j_{n_j,n_{j-1}} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_{n_{j-1}} \end{pmatrix} - \begin{pmatrix} b^j_1 \\ \vdots \\ b^j_{n_j} \end{pmatrix} \right) \tag{4.2}$$

las cuales nos van a servir para transitar de una capa a otra, esta vez no por la función de activación, si no por su derivada. Continuamos con la siguiente definición:

$$\dot{\sigma}_j(x_t) = \sigma_j' \circ \sigma_{j-1} \circ ... \circ \sigma_1(x_t) \in \mathbb{R}^{n_j} \tag{4.3}$$

Ahora vamos a dar una fórmula recursiva sin tener en cuenta los umbrales, ya que la derivada parcial respecto a un umbral (que siempre nos va a quedar $-1$) la hemos considerado al definir los $\phi_j$. Así, las matrices $W_j$ serán de dimensión $n_j \times n_{j-1}$ en vez de $n_j \times (n_{j-1}+1)$. $j \in \{1, ..., l\}$.

$$\begin{cases} \Psi_l(x_t) = \dot{\Sigma}_l(x_t) \in M_{n_l \times n_l}(\mathbb{R}) \\ \Psi_j(x_t) = \dot{\Sigma}_j(x_t) \cdot W_{j+1}^T \cdot \Psi_{j+1}(x_t) \in M_{n_j \times n_l}(\mathbb{R}) \end{cases} \tag{4.4}$$

donde para cada $j \in \{1, ..., l\}$:

$$\dot{\Sigma}_j(x_t) = diag(\dot{\sigma}_j(x_t)) \in M_{n_j \times n_j} \tag{4.5}$$

y el operador $diag : \mathbb{R}^n \to M_{n\times n}(\mathbb{R})$ simplemente transforma un vector en una matriz diagonal. Con esta notación, para la entrada $x_t \in \mathbb{R}^{n_0}$ vamos a poder calcular la matriz jacobiana en función de todos los pesos y umbrales de nuestra función de salia $F(\hat{W}, x_t) \in \mathbb{R}^{n_l}$ (notar que ahora estamos considerando $\hat{W}$). Esta matriz va a tener dimensiones $N \times n_l$.

$$J_F(\hat{W}, x_t) = \begin{bmatrix} \Psi_l(x_t) \otimes \phi_{l-1}(x_t) \\ \vdots \\ \Psi_1(x_t) \otimes \phi_0(x_t) \end{bmatrix} \tag{4.6}$$

Vamos a pararnos un poco a comprobar que en efecto, es una matriz de dimensiones $N \times n_l$. Para cada $j \in \{1, ..., l\}$:

$$dim(\Psi_j(x_t) \otimes \phi_{j-1}(x_t)) = n_j \cdot (n_{j-1} + 1) \times n_l$$

tenemos $n_l$ columnas, el número de filas lo podemos calcular sumando todas las filas de nuestros bloques:

$$\sum_{i=1}^{l} (n_{i-1} + 1) \cdot n_i = dim(\mathcal{W}) = N$$

Con esta estructuración que hemos hecho para un $x_t$ del conjunto de entrenamiento arbitrario, definimos la siguiente matriz por bloques:

$$\mathcal{P}(\hat{W}) = \left[ J_F(\hat{W}, x_1), \ldots, J_F(\hat{W}, x_T) \right] \tag{4.7}$$

sus dimensiones van a ser de $N$ filas y $T \cdot n_l$ columnas, $\mathcal{P}(\hat{W}) \in M_{N \times T \cdot n_l}(\mathbb{R})$. Vamos a recordar la fórmula de nuestra función de error:

$$\mathcal{J}(W) = \sum_{t=1}^{T} \mathcal{E}\left(F\left(W, x_t\right), g^*(x_t)\right)$$

para cada $x_t$, denotamos:

$$\nabla_{\mathcal{E}}(x_t) := \nabla \mathcal{E}\left(F\left(W, x_t\right), g^*(x_t)\right) \in \mathbb{R}^{n_l}$$

siendo $\nabla\mathcal{E}(\cdot, \cdot)$ el gradiente de $\mathcal{E}$ respecto a la primera componente. Recordar que nuestra función $\mathcal{E} : \mathbb{R}^{n_l} \times \mathbb{R}^{n_l} \to [0, +\infty)$ medía el error en cada entrada. Por último, vamos a concatenar todos nuestros vectores y finalmente definir:

$$\mathcal{G}(\hat{W}) := (\nabla_{\mathcal{E}}(x_1), ..., \nabla_{\mathcal{E}}(x_T)) \in \mathbb{R}^{T \cdot n_l} \tag{4.8}$$

de esta manera podemos caracterizar nuestro $\nabla \hat{J}(W) \in \mathbb{R}^N$ de la siguiente forma:

$$\hat{\nabla} J(W) = \mathcal{P}(\hat{W}) \cdot \mathcal{G}(\hat{W}) \in \mathbb{R}^N \tag{4.9}$$

estamos multiplicando una matriz $\mathcal{P}(\hat{W})$ por un vector $\mathcal{G}(\hat{W})$, por lo que hemos conseguido caracterizar $\nabla \hat{\mathcal{J}}(W)$ como un sistema de ecuaciones. Ahora podemos jugar con el rango de nuestra matriz $\mathcal{P}(\hat{W})$. Por ejemplo, si nuestra matriz tiene rango máximo para todo $W \in \mathcal{W}$, entonces $\nabla J(W)$ será cero solamente si $\mathcal{G}(\hat{W})$ es cero.

**Teorema 4.2.** *Sea $g^* : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ y sea $(n_0, ..., n_l)$ una arquitectura hacia delante. Tomamos un conjunto de entrenamiento $\{(x_t, g^*(x_t))\}_{t \leq T}$ y una función de error $\mathcal{J}(W)$. Supongamos que se satisface la siguiente condición:*

- *Nuestra función $\mathcal{E}(\cdot, \cdot)$ es diferenciable respecto a su primer argumento y si $\nabla \mathcal{E}(x^*, y) = \bar{0} \in \mathbb{R}^{n_l}$, entonces $x^*$ es un mínimo global de $\mathcal{E}$.*

*Si el rango de las columnas de nuestra matriz $\mathcal{P}(\hat{W})$ es máximo (igual a $T \times n_l$) $\forall\ W \in \mathcal{W}$:*

*a) Si existe $W^* \in \mathcal{W}$ tal que $F(W^*, x_t) = g^*(x_t)$ para todo $x_t$ en el conjunto de entrenamiento (aproximación exacta), entonces $W^*$ es el mínimo global de $\mathcal{J}$ y todos los puntos críticos de $\mathcal{J}$ son mínimos globales.*

*b) Si no hay aproximación exacta, entonces nuestra función $\mathcal{J}$ no tiene puntos críticos.*

*Demostración.*

Se deduce de todo lo comentado anteriormente. Al plantear $\nabla \hat{J}(W)$ como un sistema de ecuaciones, los puntos críticos serán los $\hat{W}$ verificando:

$$\mathcal{P}(\hat{W}) \cdot \mathcal{G}(\hat{W}) = 0$$

pero si $\mathcal{P}(\hat{W})$ tiene rango máximo para todos los $W$, entonces, en el caso de que exista, la única solución es la trivial $\mathcal{G}(\hat{W}) = \bar{0}$. Si existe estamos en el caso $a$) y si no existe estamos en el caso $b$).

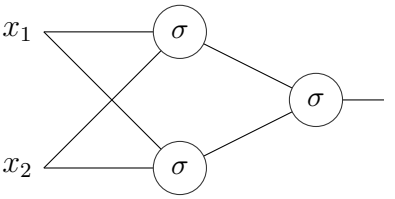
□

La condición que le imponemos a nuestra función $\mathcal{E}$ no es tan restrictiva como parece ya que podemos elegirla de tal forma. Por ejemplo, podemos tomar $\mathcal{E}(x, y) = \|x - y\|_2^2$. De este teorema se deduce también una estimación del tamaño de nuestra red. Esta claro que conforme $T$ sea más grande (tengamos más entradas en el conjunto de entrenamiento), necesitaremos de un modelo más grande.

Nuestra matriz $\mathcal{P}(\hat{W})$ tiene $T \cdot n_l$ columnas que podemos verlas como vectores en $\mathbb{R}^N$. Si queremos que tenga rango máximo, esas columnas deberán ser linealmente independientes para todo $W \in \mathcal{W}$, por lo que una cota necesaria para que se cumpla el teorema es:

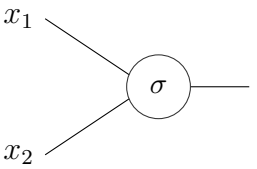
$$T \cdot n_l \leq N$$

por ejemplo, es imposible que 5 vectores en $\mathbb{R}^3$ sean linealmente independientes. En [12] se puede encontrar un estudio de cómo estimar el tamaño de la red. Una condición que puede resultar difícil de comprobar es la de existencia de una aproximación exacta. También, hay muchos modelos que son muy buenos aproximando una función sin necesidad de tener una aproximación exacta, por ejemplo, en el Ejemplo 3.5. nos aproximábamos a la función lógica $XOR$ con la siguiente arquitectura:




tomando unos pesos iniciales adecuados obteníamos un error tan pequeño como nosotros quisiéramos. Como $0 < \sigma(x) < 1$; $\forall x \in \mathbb{R}$ no existe aproximación exacta. No obstante, aunque no existan unos pesos que nos garanticen un mínimo global, la condición de rango máximo sigue siendo útil. Supongamos que no existe aproximación exacta pero se cumple que el rango de $\mathcal{P}(\hat{W})$ es máximo, entonces estamos en el caso $b$) del teorema 4.2. No hay puntos críticos de la función error pero eso no implica que el error no pueda ser tan pequeño como nosotros queramos. Después de el ejemplo 3.5. explicábamos este fenómeno.

**Ejemplo 4.1.** *Vamos a comprobar la condición de rango para una sola neurona, entradas en* $\mathbb{R}^2$ *y salidas en* $\mathbb{R}$*. Notar que lo podemos ver como una arquitectura* $(2, 1)$*.*




*La salida de la neurona va a ser el valor* $\sigma(x_1 \cdot w_1 + x_2 \cdot w_2 - b)$*. Con* $\sigma$ *la función logística.*

$$\sigma(x) = \frac{1}{1+e^{-x}}; \quad \sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

$$\Psi_1(x) = \sigma'(x_1 \cdot w_1 + x_2 \cdot w_2 - b); \quad \phi_0(x) = (x_1, x_2, -1)$$

$$J_F(\hat{W}, x) = \Big[\Psi_1(x) \otimes \phi_0(x)\Big] = \begin{bmatrix} x_1 \cdot \sigma'(x_1 \cdot w_1 + x_2 \cdot w_2 - b) \\ x_2 \cdot \sigma'(x_1 \cdot w_1 + x_2 \cdot w_2 - b) \\ -\sigma'(x_1 \cdot w_1 + x_2 \cdot w_2 - b) \end{bmatrix} \in M_{3\times 1}(\mathbb{R}) \simeq \mathbb{R}^3$$

*Aquí se ve claramente que la matriz $J_F(\hat{W}, x)$ recoge todas las derivadas parciales. Ahora supongamos que de una hipotética $g^* : \mathbb{R}^2 \to \mathbb{R}$ solamente sabemos que $g^*((0,0)) = 1/2$, $g^*((1,0)) = 1/4$ y $= g^*((0,1)) = 1/8$. Para esas tres entradas podemos construir la matriz $\mathcal{P}(\hat{W})$.*

$$\mathcal{P}(\hat{W}) = \begin{bmatrix} 0 & \sigma'(w_1 - b) & 0 \\ 0 & 0 & \sigma'(w_2 - b) \\ -\sigma'(-b) & -\sigma'(w_1 - b) & -\sigma'(w_2 - b) \end{bmatrix} \in M_{3\times 3}(\mathbb{R})$$

*esta matriz tiene rango máximo ya que su determinante es distinto de cero para todo $W$ (ya que $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$ es positiva).*

$$|\mathcal{P}(\hat{W})| = -\sigma'(-b) \cdot \sigma'(w_1 - b) \cdot \sigma'(w_2 - b)$$

*Ahora, veamos si existe aproximación exacta para esas entradas y salidas.*

$$\sigma(-b) = \frac{1}{2}; \quad \sigma(w_1 - b) = \frac{1}{4}; \quad \sigma(w_2 - b) = \frac{1}{8}$$

$$b = 0; \quad w_1 = -log(3); \quad w_2 = -log(7)$$

*Tomamos $\mathcal{E}(x, y) = (x - y)^2$ ya que esta función cumple las hipótesis del teorema (es convexa) y se concluye del teorema anterior que:*

$$W = \left(-log(3) \quad -log(7) \;\middle|\; 0\right)$$

*es un mínimo global de la función $\mathcal{J}$.*

Las dimensiones de la matriz $\mathcal{P}(\hat{W})$ aumentan bastante rápido y se precisaría de la ayuda de un ordenador para construirla, de ahí que hayamos optado por utilizar una sola neurona. La comprobación de que $\mathcal{P}(\hat{W})$ tiene rango máximo para todo $W \in \mathcal{W}$ también será más costosa según el tamaño aumente. Por ejemplo, aunque sabemos que en este caso no se cumplen las hipótesis del teorema, para la aproximación de la función lógica $XOR$ tendríamos que $\mathcal{P}(\hat{W})$ tiene dimensiones $9 \times 4$.

La localización de los mínimos globales en las funciones de error $\mathcal{J}$ son bastante delicadas y hoy en día se sigue investigando al respecto para intentar relajarlas un poco y tener una forma de caracterizarlos más sencilla.

En la vida real los modelos de redes neuronales que se utilizan, como veremos en el siguiente capítulo, están formados por arquitecturas bastante grandes, haciendo que sea costoso construir $\mathcal{P}(\hat{W})$ y muy difícil saber si esa matriz verifica la condición de rango máximo.

# Capítulo 5

# Entrenando una red neuronal artificial

Para terminar, en este último capítulo vamos a entrenar una red neuronal para que aprenda a distinguir qué número entre el 0 y el 9 se encuentra en una foto de $28 \times 28$ píxeles en escala de grises. Este ejemplo servirá para ilustrar lo complejos que pueden llegar a ser los problemas de la vida real y ver que hace falta todavía mucha investigación respecto a los temas que hemos tratado.

El conjunto de datos está formado por 70000 imágenes, de la cuales 60000 (85.7 %) pertenecen a el conjunto de entrenamiento y 10000 (14.3 %) a el conjunto de test.

**Figura 5.1:** Algunas de las imágenes de los números.

Estas imágenes en escala de grises son matrices de tamaño $28 \times 28$. En cada celda habrá un número entre el cero y el uno (cero y uno incluidos). Cuanto más cerca del blanco esté el píxel, más cerca del cero estará esa celda (análogo con el negro).

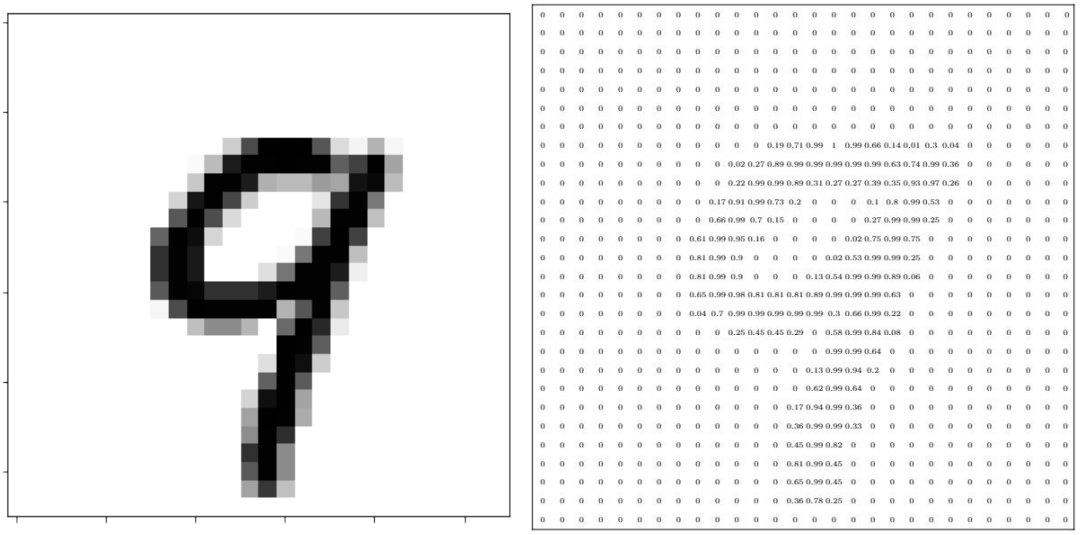


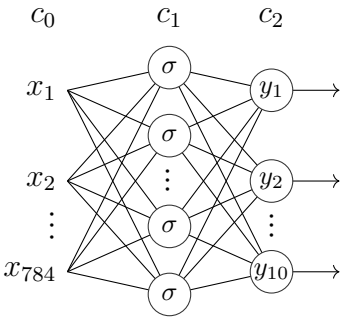
**Figura 5.2:** Representación de una foto en forma matricial.

A su vez, estas matrices las podemos transformar en vectores de dimensión $28 \cdot 28 = 784$ concatenando todas sus filas (igual que hacíamos en la Sección 4.2. con la aplicación $\hat{\cdot}$). De esta manera, tendremos que las entradas tendrán tamaño $n_0 = 784$. Como es un problema de clasificación ($n_l = 10$); para las salidas hacemos la asignación habitual:

$$\begin{aligned} \text{Foto con un } 0 &\longrightarrow (1,0,0,0,0,0,0,0,0,0) \\ &\vdots \\ \text{Foto con un } 9 &\longrightarrow (0,0,0,0,0,0,0,0,0,1) \end{aligned}$$

Vamos a utilizar una arquitectura $(784, 512, 10)$; con la función ReLU[1] en la capa oculta y la softmax para 10 clases en la salida:

$$\sigma(x) = máx\{0, x\}; \quad y_j = \frac{e^{x_j}}{\sum_{k=1}^{10} e^{x_k}}$$




Vamos a considerar el siguiente error en cada entrada:

[1]Esta función no es derivable en $x = 0$, pero podemos tomar $\sigma'(0) := 0$.

$$\mathcal{E}(x,y) = -\sum_{j=1}^{10} log(x_j)\cdot y_j$$

y como optimizador de nuestra función vamos a tomar el algoritmo de descenso por gradiente, con $\alpha = 0.2$:

$$\begin{cases} W^{(1)} = 0 \in \mathcal{W} \\ W^{(n+1)} = W^{(n)} - 0.2\nabla\mathcal{J}(W^{(n)}) \end{cases}$$

Después de iterar el algoritmo 2000 veces (2 segundos por iteración = 4000 segundos = una hora, 6 minutos y 40 segundos) obtenemos:

| | $\mathcal{J}(W^{(2000)})$ | Aciertos | Fallos | Precisión |
|---|---|---|---|---|
| Entrenamiento | 0.0837 | 58686 | 1314 | 97.81 % |
| Test | 0.1003 | 9715 | 285 | 97.15 % |

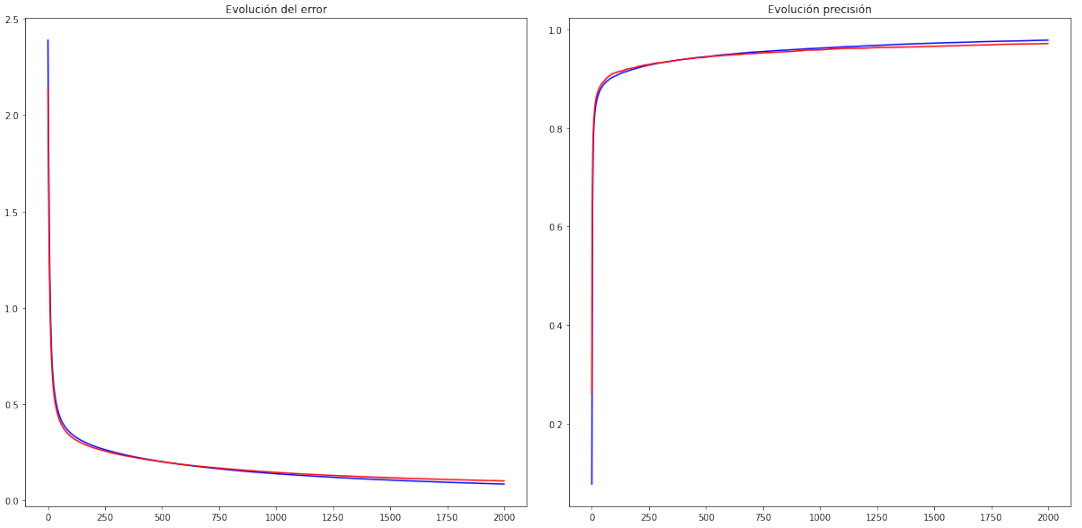



**Figura 5.3:** A la izquierda una evolución del error en el conjunto de entrenamiento (azul) y en el conjunto de test (rojo). A la derecha una evolución de la precisión.

Para medir la precisión, lo que hemos hecho es determinar la proporción de aciertos y para determinar la clase a la que pertenece cierta entrada $x$ hemos considerado la componente más grande de cada vector. Por ejemplo si:

$$F(W,x) = (0.98, 0.002, 0.001, 0.001, 0.001, 0.001, 0.001, 0.002, 0.011)$$

diremos que la foto $x$ pertenece a la clase del 0.

La implementación de esta red neuronal se ha hecho utilizando el lenguaje de programación Python y se puede acceder a el programa desde el siguiente enlace: `https://github.com/gosantam/EjemploTFG`.

Vemos que la mayoría del progreso se obtiene en las primeras iteraciones de el algoritmo. Se pueden encontrar mejores modelos y formas más rápidas de entrenar este modelo pero no vamos a detenernos más en esto.

En este problema en concreto, con el Teorema 4.2. no podemos asegurar la convergencia hacia un mínimo global ya que:

$$T \cdot n_2 = 60000 \cdot 10 = 600000 \geq 407050 = (784 + 1) \cdot 512 + (512 + 1) \cdot 10 = N$$

y debería darse justo la desigualdad contraria $T \cdot n_2 \leq N$. Tampoco se cumple la condición de existencia de una aproximación exacta, el motivo es el mismo que explicamos anteriormente para la función lógica $XOR$.

$$0 < y_j(x) < 1 \ \ \forall x \in \mathbb{R}^{10}$$

Al plantear el sistema:

$$\mathcal{P}(\hat{W}) \cdot \mathcal{G}(\hat{W}) = \bar{0} \in \mathbb{R}^N$$

podría haber soluciones distintas de la solución trivial $\mathcal{G}(\hat{W}) = \bar{0} \in \mathbb{R}^{600000}$. Aun así, vemos que estas redes tienen un tamaño muy grande y nuestra matriz $\mathcal{P}(\hat{W})$ tiene dimensiones $407050 \times 600000$. En matrices de tanta dimensión determinar el rango de sus columnas (además para cada $W$ distinto) es una tarea ingente.

# Conclusiones

En este último capítulo, hemos conseguido construir un modelo que es capaz de decirnos con bastante fiabilidad qué número se encuentra en una fotografía con algún dígito entre el cero y el nueve.

Acabamos de ver que esto no se ha conseguido por arte de magia y que, una vez más, las matemáticas están detrás para explicarnos con todo el detalle lo que está ocurriendo. En mi opinión, todavía quedan muchos cabos sueltos y muchas preguntas por responder, no solo en los temas que hemos tratado, sino también (y en mayor medida) en modelos de redes neuronales más complejos.

A pesar de estos agujeros, pienso que el tiempo que utilizamos en comprender las cosas de forma matemática, aunque nos lleve un gran esfuerzo de abstracción, nos sirve a todos para entender mejor el mundo que nos rodea, y más aun cuando hablamos sobre una creación humana como lo es la informática.

# Bibliografía


[1] Manuel Bello Hernández. *Apuntes de análisis real y funcional.* Universidad de La Rioja, Curso 2019-2020.

[2] Fjodor van Veen. *The Neural Network Zoo (2016).* The Asimov Institute, disponible en: `https://www.asimovinstitute.org/neural-network-zoo/`.

[3] Fjodor van Veen. *Neural Network Zoo Prequel: Cells and Layers (2017).* The Asimov Institute, disponible en: `https://www.asimovinstitute.org/author/fjodorvanveen/`.

[4] Haim Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations.* Universitext, Springer, 2010.

[5] Leonardo Ferreira Guilhoto. An overview of artificial neural networks for mathematicians. disponible en: `http://math.uchicago.edu/~may/REU2018/REUPapers/Guilhoto.pdf`.

[6] Luz Gloria Torres. *Redes neuronales y aproximación de funciones.* Universidad Nacional de Colombia, disponible en: `https://dialnet.unirioja.es/descarga/articulo/6953010.pdf`.

[7] Sebastian Ruder. *An overview of gradient descent optimization algorithms (2016).* disponible en: `https://ruder.io/optimizing-gradient-descent/`.

[8] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. disponible en: `http://www.cs.toronto.edu/~fritz/absps/momentum.pdf`.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. disponible en: `http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf`.

[10] Edwing K.P Chong y Stanislaw H. Zak. *An introduction to optimization, second edition.*

[11] Rubén Mazo tomás. *Fundamentación matemática del aprendizaje automático y profundo.* Universidad de La Rioja, Curso 2017-2018.

[12] Hao Shen. Towards a mathematical understanding of the difficulty in learning with feedforward neural networks (2017). disponible en: https://arxiv.org/pdf/1611.05827v3.pdf.

[13] Hao Shen. A differential topological view of challenges in learning with feedforward neural networks (2018). disponible en: https://arxiv.org/pdf/1811.10304v1.pdf.